



# 빌드 및 배포 방법

☀ Status	완료
☰ Epic	최종 산출물
📅 Starting date	@2023년 4월 17일
📅 Deadline	@2023년 5월 19일



## Gitlab 소스 클론 이후 빌드 및 배포할 수 있도록 정리한 문서

- 1) 사용한 JVM, 웹서버, WAS 제품 등의 종류와 설정값, 버전(IDE 버전 포함) 기재
- 2) 빌드 시 사용되는 환경 변수 등의 주요 내용 상세 기재
- 3) 배포 시 특이사항 기재
- 4) DB 접속 정보 등 프로젝트(ERD)에 활용되는 주요 계정 및 프로퍼티가 정의된 파일 목록

## 환경 정보

### 하드웨어 정보

```
- Architecture: x86_64
- CPU Model: Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz
- CPU(s): 4
- CPU MHz: 2300.030
- CPU family: 6
- CPU op-mode(s): 32-bit, 64-bit
- Core(s) per socket: 4
- Socket(s): 1
- Thread(s) per core: 1
```

### 소프트웨어 정보

```
- OS : Ubuntu 20.04.6 LTS (Focal Fossa)
- 도커 버전 : Docker version 23.0.5
- 패키지 관리자 : apt 2.0.9 (amd64)
```

## 기술 스택 & 개발 환경

### Backend - Spring

```
- Java openjdk-11
- Spring boot 2.7.1
- Spring Security 2.7.10
- Spring Data JPA 2.7.10
- Querydsl 5.0.0
- JWT 0.11.2
- Spring Data Redis 2.7.10
```

## Frontend - Next.js

```
- Node.js 18.16.0
- React 18.2.0
- Next.js 13.2.4
- TypeScript 5.0.4
- Styled Components 5.3.9
- Tanstack Query 4.29.5
- Zustand 4.3.7
- React Hook Form 7.43.9
- Zod 3.21.4
- ESLint 8.38.0
- Prettier 2.8.7
- Axios 1.3.6
- Monaco Editor 0.38.0
- Tiptap Editor 2.0.3
```

## CI / CD

```
- AWS EC2 / Ubuntu 20.04.6 LTS (Focal Fossa)
- Docker 23.0.5
- Docker Compose 1.25.0
- SSL (certbot) 1.1.1f
- Jenkins 24.0.2
- NGINX 1.23.4
- sonarqube 9.9.1.69595
- pmd 3.4.0
- checkstyle 10.9.3
```

## OAuth2.0 소셜 로그인

### application.yml

```
spring:
  security:
    oauth2:
      client:
        registration:
          github:
            client-id: ${OAUTH2_CLIENT_ID}
            client-secret: ${OAUTH2_CLIENT_SECRET}
            redirect-uri: "{baseScheme}://{baseHost}/api/v1/login/oauth2/code/{registrationId}"
```

## Github Login

Settings > Developer settings > OAuth Apps

Homepage URL \*

The full URL to your application homepage.

Application description

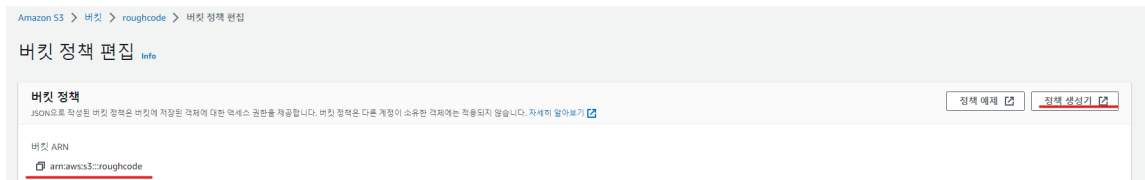
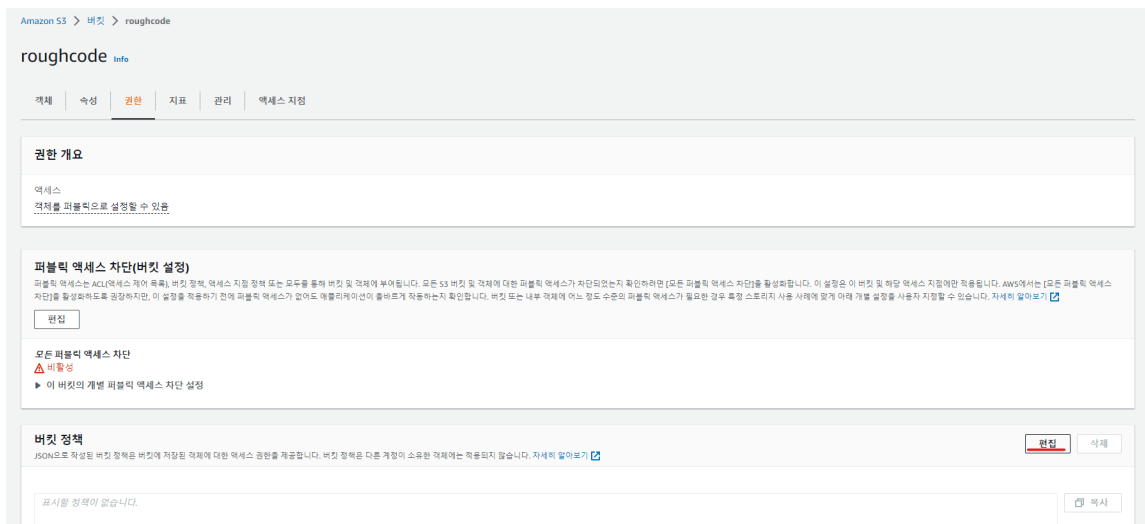
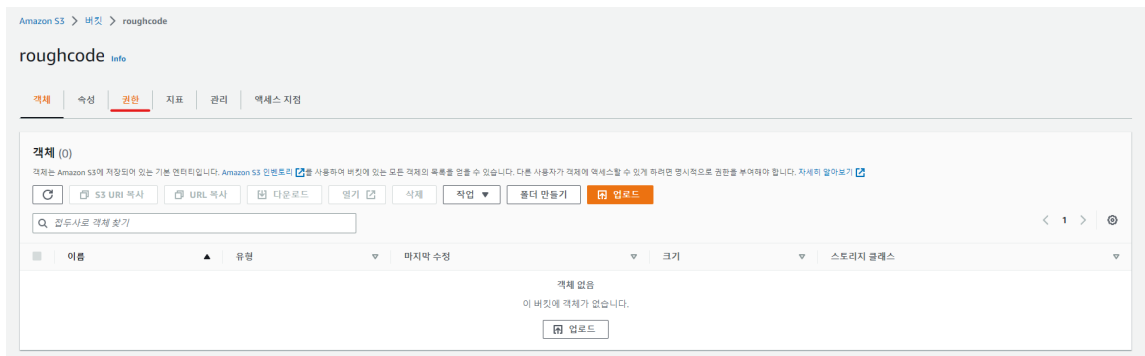
개발자 지망생과 신입 개발자를  
위한 프로젝트 공유 및 코드 리뷰

This is displayed to all users of your application.

Authorization callback URL \*

Your application's callback URL. Read our [OAuth documentation](#) for more information.

## 버킷 설정



- Action에는 `GetObject`, `PutObject`, `DeleteObject` 3개를 체크하고 ARN에는 복사해둔 ARN값을 입력한다.
- ARN값을 입력하되 `/*` 값도 추가 해줘야한다.



## AWS Policy Generator

The AWS Policy Generator is a tool that enables you to create policies that control access to Amazon Web Services (AWS) products and resources. For more information about creating policies, see [key concepts in Using AWS Identity and Access Management](#). Here are [sample policies](#).

### Step 1: Select Policy Type

A Policy is a container for permissions. The different types of policies you can create are an IAM Policy, an S3 Bucket Policy, an SNS Topic Policy, a VPC Endpoint Policy, and an SQS Queue Policy.

Select Type of Policy S3 Bucket Policy

### Step 2: Add Statement(s)

A statement is the formal description of a single permission. See [a description of elements](#) that you can use in statements.

Effect ☒ Allow ☐ Deny

Principal \*

Use a comma to separate multiple values.

AWS Service

Amazon S3

☐ All Services (\*)

Use multiple statements to add permissions for more than one service.

Actions 3 Action(s) Selected

☐ All Actions (\*)

Amazon Resource Name (ARN)

arn:aws:s3:::roughcode/\*

ARN should follow the following format: arn:aws:s3:::{BucketName}/{Key\*Name}.  
Use a comma to separate multiple values.

[Add Conditions \(Optional\)](#)

[Add Statement](#)

Amazon S3 > 버킷 > roughcode > 버킷 정책 편집

### 버킷 정책 편집

버킷 정책  
JSON으로 작성된 버킷 정책은 버킷에 지정된 객체에 대한 액세스 권한을 제공합니다. 버킷 정책은 다른 계정이 소유한 객체에는 적용되지 않습니다. 자세히 알아보기

버킷 ARN  
arn:aws:s3:::roughcode

정책

```
1 {
2   "Id": "Policy1682036652653",
3   "Version": "2012-10-17",
4   "Statement": [
5     {
6       "Sid": "Stmt1682036652653",
7       "Action": [
8         "s3:DeleteObject",
9         "s3:GetObject",
10        "s3:PutObject"
11      ],
12       "Effect": "Allow",
13       "Resource": "arn:aws:s3:::roughcode/*",
14       "Principal": "*"
15     }
16   ]
17 }
```

문 편집

문 선택  
정책에서 기존 문을 선택하거나 새 문을 추가합니다.  
[+ 새 문 추가](#)

## 사용자 설정

AWS > 서비스 > IAM

'IAM'에 대한 검색 결과

서비스 (8)  
특징 (19)  
리소스 (61)  
블로그 (61)  
설명서 (2,554)  
자서 기사 (15)  
마켓플레이스 (464)

서비스

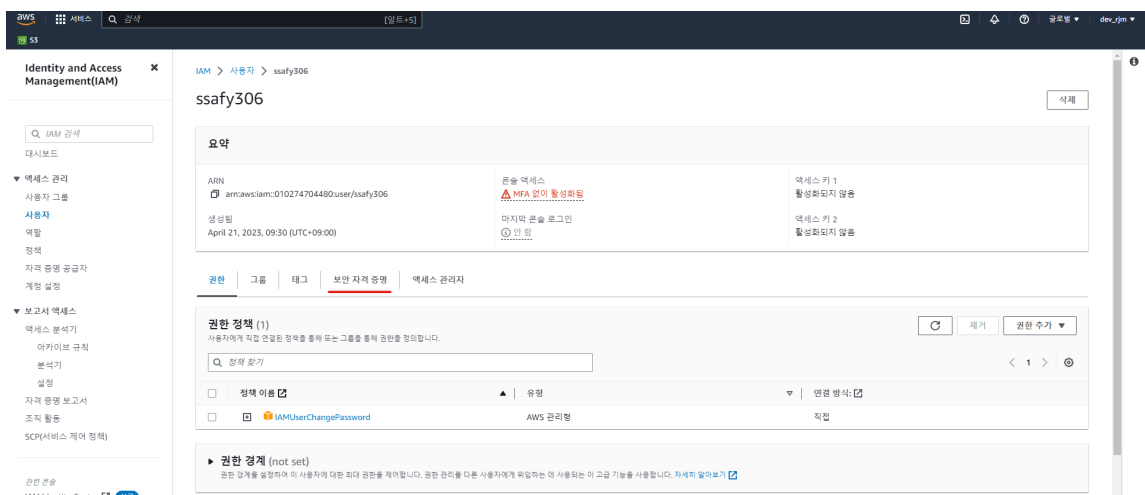
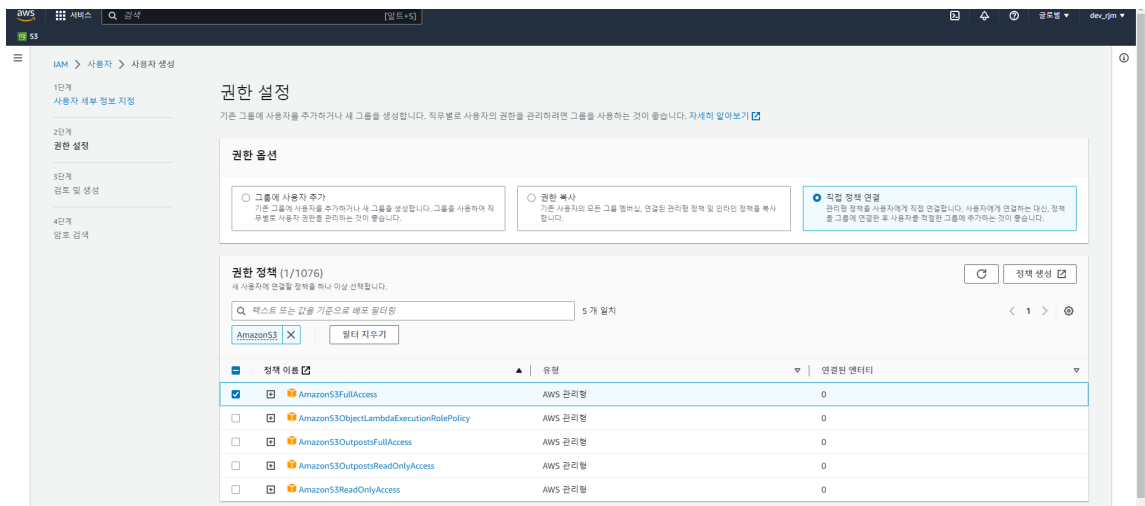
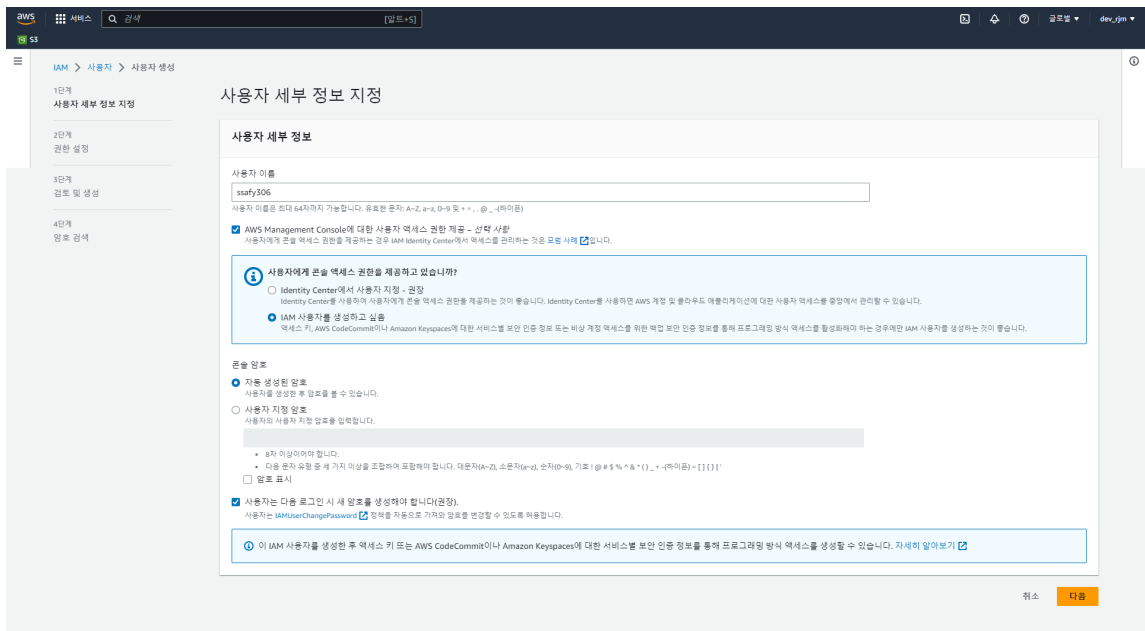
- IAM ☆  
AWS 리소스에 대한 액세스 관리  
주요 기능  
그룹 사용자 역할 정책 액세스 분석기
- IAM Identity Center(AWS Single Sign-On의 후속 서비스) ☆  
여러 AWS 계정 및 클라우드 애플리케이션에 대한 업무 환경 사용자 액세스 관리

AWS > 서비스 > IAM > 사용자

Identity and Access Management(IAM)

사용자 (0) 정보  
IAM 사용자는 계정에서 AWS와 상호 작용하는 데 사용되는 장기 자격 증명을 가진 자격 증명입니다.  
[사용자 이름 또는 액세스 키로 사용자 찾기](#)

사용자 이름	그룹	마지막 활동	MFA	암호 수명	활성 키 수명
표시할 리소스 없음					





1단계

## 액세스 키 모범 사례 및 대안

2단계 - 선택 사항

설정 태그 설정

3단계

액세스 키 검색

## 액세스 키 모범 사례 및 대안

보안 개선을 위해 액세스 키와 같은 장기 자격 증명을 사용하지 마세요. 다음과 같은 사용 사례와 대안을 고려하세요.

☐ Command Line Interface(CLI)

AWS CLI를 사용하여 AWS 계정의 액세스할 수 있도록 이 액세스 키를 사용할 것입니다.

☐ 로컬 코드

로컬 개발 환경의 애플리케이션 코드를 사용하여 AWS 계정에 액세스할 수 있도록 이 액세스 키를 사용할 것입니다.

☐ AWS 클라우드 서비스에서 실행되는 애플리케이션

Amazon EC2, Amazon ECS 또는 AWS Lambda와 같은 AWS 클라우드 서비스에서 실행되는 애플리케이션 코드를 사용하여 AWS 계정에 액세스할 수 있도록 이 액세스 키를 사용할 것입니다.

☐ 서버 패티 서비스

AWS 리소스를 모니터링 또는 관리하는 서버 패티 애플리케이션 또는 서비스에 액세스할 수 있도록 이 액세스 키를 사용할 것입니다.

☐ AWS 외부에서 실행되는 애플리케이션

애플리케이션을 온프레미스 호스트에서 실행하거나 로컬 AWS 클라이언트 또는 서버 패티 AWS 클라이언트를 사용할 수 있도록 이 액세스 키를 사용할 것입니다.

☒ 기타

귀하의 사용 사례가 여기에 나열되어 있지 않습니다.



이 사용 사례에서는 액세스 키를 사용해도 되지만 모범 사례를 따릅니다.

• 액세스 키를 일련 텍스트, 코드 리포지토리 또는 코드포 작성하지는 않습니다.

• 더 이상 필요 없는 경우 액세스 키를 비활성화하거나 삭제합니다.

• 최소 권한을 활성화합니다.

• 액세스 키를 정기적으로 교체합니다.

액세스 키 관리에 대한 자세한 내용은 [AWS 액세스 키 관리 모범 사례](#)를 참조하세요.

원소

다음

## 수정

Amazon S3

바킷

액세스 지정

객체 Lambda 액세스 지정

다중 리전 액세스 지정

백지 작업

S3용 IAM Access Analyzer

이 계정의 퍼블릭 액세스 차단 설정

## ▼ Storage Lens

대시보드

AWS Organizations 설정

가능 스토프라이트

▶ S3용 AWS Marketplace

Amazon S3 &gt; 바킷 &gt; roughcode &gt; 객체 소유권 편집

## 객체 소유권 편집

Info

## 객체 소유권

다른 AWS 계정에서 이 바킷에 작성한 객체의 소유권 및 액세스 제어 목록(ACL)의 사용을 제어합니다. 객체 소유권은 객체에 대한 액세스를 지정할 수 있는 사용자를 결정합니다.

☐ ACL 비활성화됨(권장)

이 바킷의 모든 객체는 이 계정이 소유합니다. 이 바킷과 그 객체에 대한 액세스는 정책을 통해서만 제공됩니다.

☒ ACL 활성화됨

이 바킷의 객체는 다른 AWS 계정에서 소유할 수 있습니다. 이 바킷 및 객체에 대한 액세스는 ACL을 사용하여 지정할 수 있습니다.



각 객체의 액세스를 개별적으로 제어하거나 객체 라우터가 업로드하는 데이터를 소유자로 하여 주는 경우가 아니라면 ACL을 비활성화하는 것이 좋습니다. ACL 대신 바킷 정책을 사용하여 계정 외부의 사용자와 데이터를 공유하면 권한을 관리하고 감사하기가 용이합니다.

## 객체 소유권

☐ 바킷 소유자 선택

이 바킷에 작성된 새 객체가 bucket-owner-full-control ACL을 지칭하는 경우 새 객체는 바킷 소유자가 소유됩니다. 그렇지 않은 경우 객체 라우터가 소유됩니다.

☒ 객체 라우터

객체 라우터는 객체 소유자로 유지됩니다.

원소

변경 사항 저장

Amazon S3

바킷

액세스 지정

객체 Lambda 액세스 지정

다중 리전 액세스 지정

백지 작업

S3용 IAM Access Analyzer

이 계정의 퍼블릭 액세스 차단 설정

## ▼ Storage Lens

대시보드

AWS Organizations 설정

가능 스토프라이트

▶ S3용 AWS Marketplace

Amazon S3 &gt; 바킷 &gt; roughcode

## roughcode

Info

객체

속성

권한

지표

관리

액세스 지정

## 권한 개요

액세스

이 계정의 권한 부여된 사용자만

## 퍼블릭 액세스 차단(바킷 설정)

퍼블릭 액세스는 ACL(액세스 제어 목록), 바킷 정책, 액세스 지정 정책 또는 모두를 통해 바킷 및 객체에 부여됩니다. 모든 S3 바킷 및 객체에 대한 퍼블릭 액세스가 차단되었는지 확인하려면 (또는 퍼블릭 액세스 차단)을 활성화합니다. 이 설정은 이 바킷 및 해당 액세스 지정에만 적용됩니다. AWS에서는 (공통 이름) 액세스 차단을 활성화하도록 권장하지만, 이 설정을 적용하기 전에 퍼블릭 액세스가 없이도 애플리케이션이 올바르게 작동하는지 확인합니다. 바킷 또는 내부 객체에 어느 정도 수준의 퍼블릭 액세스가 필요한 경우 해당 스토리지 사용 사례에 맞게 아래 가능 설정을 사용자 지정할 수 있습니다. 자세히 알아보기

편집

## 공통 퍼블릭 액세스 차단



▶ 이 바킷의 개별 퍼블릭 액세스 차단 설정

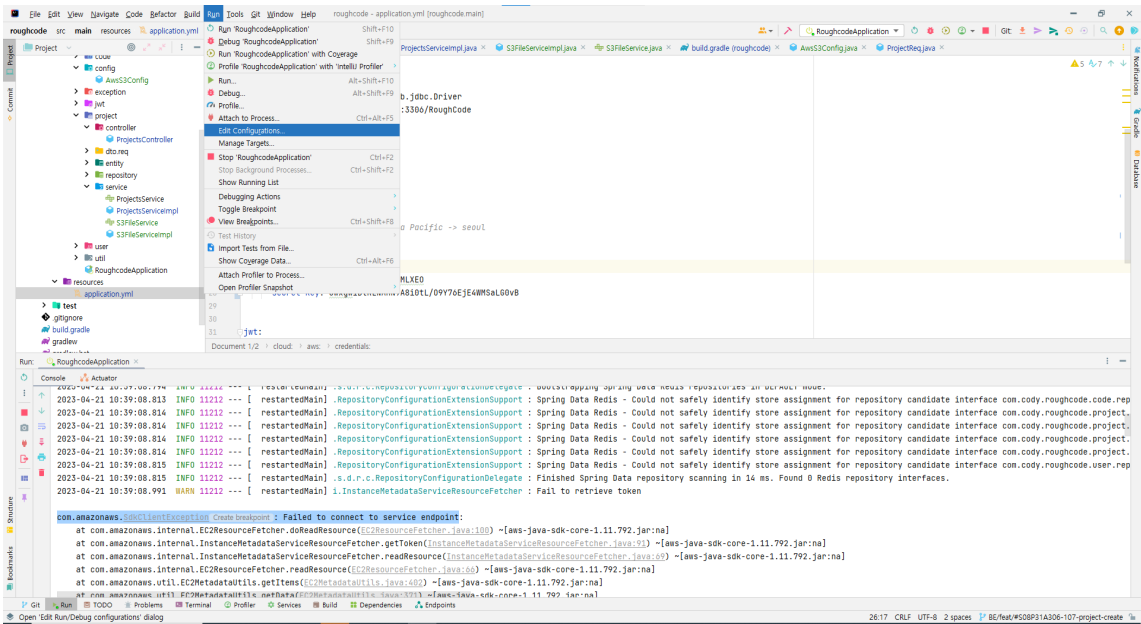
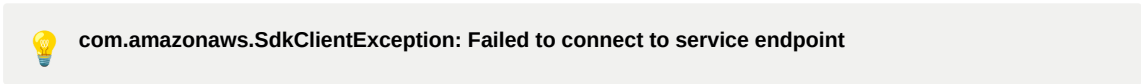


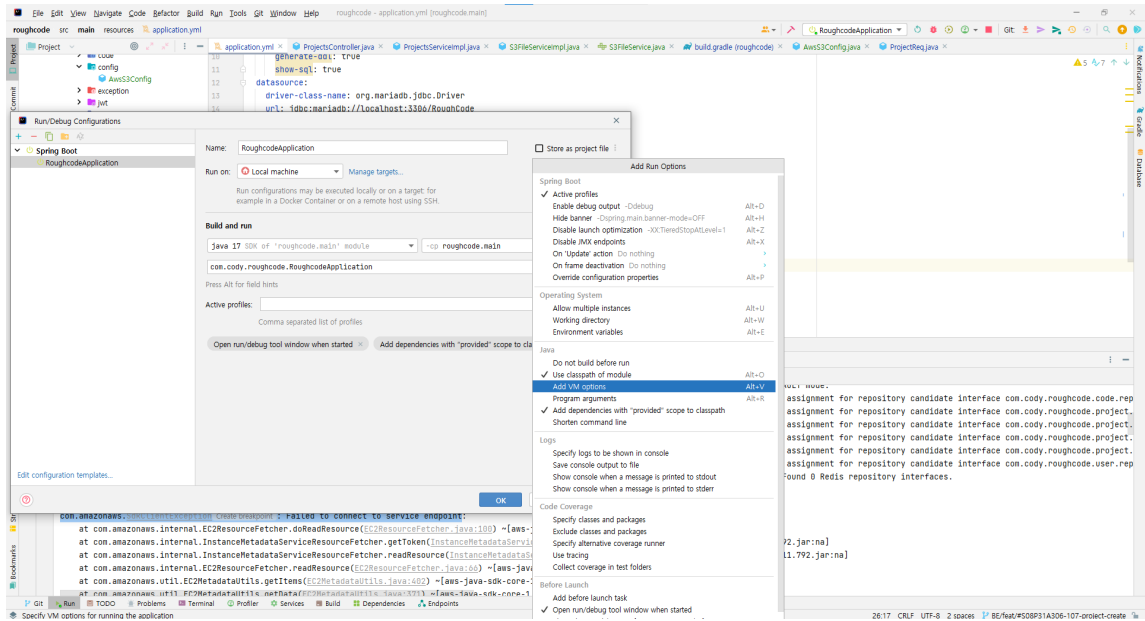


## 발급 받은 key 등록 - SpringBoot application.yml

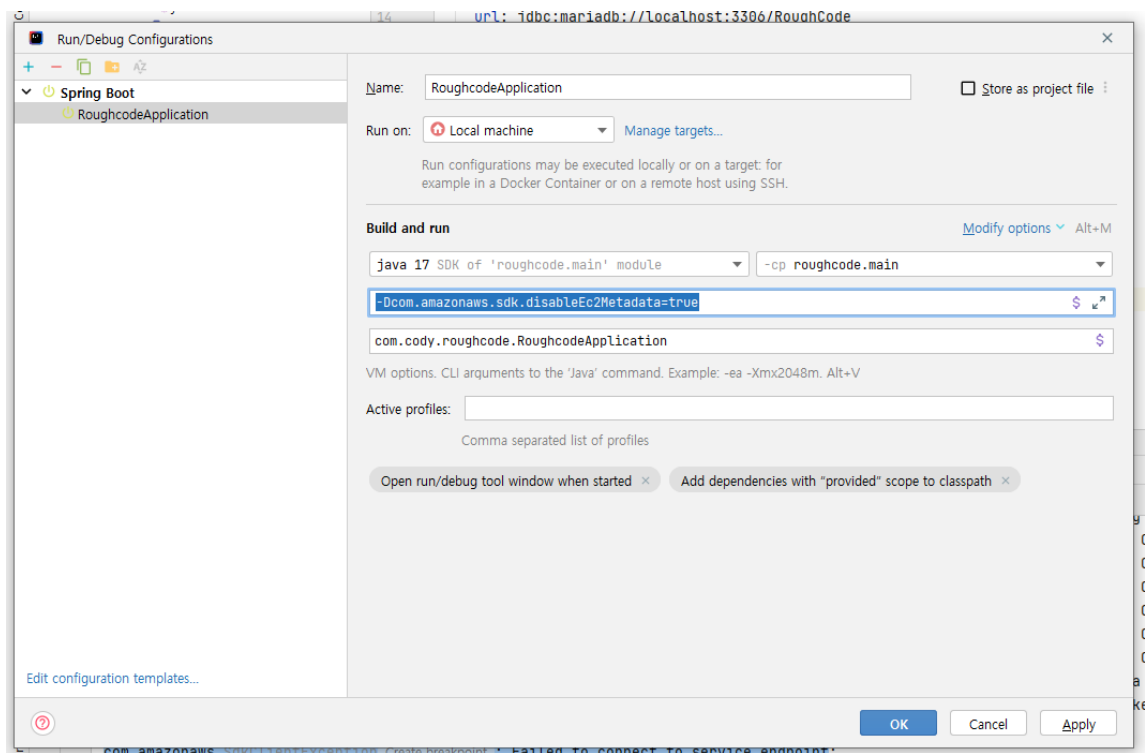
```
cloud:
  aws:
    s3:
      bucket: 버킷 이름
      region:
        static: ap-northeast-2 #Asia Pacific -> seoul
      stack:
        auto: false
      credentials:
        access-key: S3 사용자 access-key
        secret-key: S3 사용자 secret-key
```

## ▼ S3 오류해결





- VM Option 등록 : `-Dcom.amazonaws.sdk.disableEc2Metadata=true`



## application.yml

```
logging:
  level:
    com:
      amazonaws:
        util:
          EC2MetadataUtils: error
```

## ▼ CloudFront 연결

### CloudFront 검색 -> 배포 생성

CloudFront > 배포 > 생성

#### 배포 생성

**원본**

원본 도메인  
AWS 원본을 선택하거나 사용자 원본의 도메인 이름을 입력합니다.

원본 경로 - 선택 사항 **정보**  
원본 요청의 원본 도메인 이름에 추가될 URL 경로를 입력합니다.

이름  
이 원본의 이름을 입력합니다.

원본 액세스 **정보**

☐ 공개  
비밀은 공개 액세스를 허용하여 합니다.

☒ 원본 액세스 제어 설정(권장)  
비밀은 CloudFront에 대한 액세스를 제한할 수 있습니다.

☐ Legacy access identities  
CloudFront 원본 액세스 ID(OAI)를 사용하여 S3 버킷에 액세스합니다.

Origin access control  
Select an existing origin access control (recommended) or create a new configuration.

제어 설정 생성

비밀 정책  
Policy must allow access to CloudFront IAM service principal role.

☒ 정책을 수동으로 업데이트하십시오

#### 제어 설정 생성

이름

이름은 고유해야 합니다. 문자, 숫자 및 대부분의 특수 문자를 사용할 수 있습니다. 최대 64자까지 사용할 수 있습니다.

설명 - 선택 사항

설명은 최대 256자까지 입력할 수 있습니다.

서명 동작

☐ 요청 서명 안 함

☒ 서명 요청 (권장)

☐ 승인 헤더 재정의 안 함  
수신 요청에 승인 헤더가 있는 경우 서명하지 마세요.

원본 유형

원본 유형은 원본 도메인과 동일한 유형이어야 합니다.

취소 생성

[illegible]

취소 다음

출시 발표

이제 ACM에서 타원곡선 DSA(ECDSDA)인증서를 발급할 수 있습니다. 자세한 알아보고 [여러분의 생각에 대해 알려주세요.](#)

모든 인증서 보기

acs:ListCertificates가 여러 각 인증서 요약에서 세부 정보를 지원합니다. 인증서 테이블에서 페이지당 최대 500개의 인증서를 볼 수도 있습니다. 인증서 목록 페이지에서 기타 모양 아이콘을 클릭하여 테이블 기본 설정을 지정합니다. 자세한 알아보고 [여러분의 생각에 대해 알려주세요.](#)

AWS Certificate Manager > 인증서 > 인증서 요청 > 퍼블릭 인증서 요청

퍼블릭 인증서 요청

도메인 이름

인증서에 대해 하나 이상의 도메인 이름을 제공합니다.

완전히 정규화된 도메인 이름 [정보](#)

rough-code.com

이 인증서에 다른 이름 추가

이 인증서에 이름을 추가할 수 있습니다. 예를 들어, 'www.example.com'에 대한 인증서를 요청하는 경우 고객이 두 이름 중 하나로 사이트의 접속할 수 있도록 'example.com'이라는 이름을 추가할 수 있습니다.

검증 방법 [정보](#)

도메인 소유권을 검증하기 위한 방법 선택

☒ DNS 검증 - 권장

인증서 요청에서 도메인에 대한 DNS 구성을 수정할 권한이 있는 경우 이 옵션을 선택합니다.

☐ 이메일 검증

인증서 요청에서 도메인에 대한 DNS 구성을 수정할 권한을 소유하지 않거나 획득할 수 있는 경우 이 옵션을 선택합니다.

키 알고리즘 [정보](#)

암호화 알고리즘을 선택합니다. 일부 알고리즘은 일부 AWS 서비스에서 지원되지 않을 수 있습니다.

☒ RSA 2048

RSA는 가장 널리 사용되는 키 유형입니다.

☐ ECDSA P256

암호화 강도는 RSA 3072와 동일합니다.

☐ ECDSA P384

암호화 강도는 RSA 7680과 동일합니다.

태그 [정보](#)

인증서를 쉽게 관리할 수 있도록 선택적으로 각 리소스에 고유한 메타데이터를 태그 형식으로 지정할 수 있습니다.

이 리소스와 연결된 태그가 없습니다.

태그 추가

태그를 50개 더 추가할 수 있습니다.

뒤로

이전

요청

배포 도메인으로 S3 접속

## 빌드 & 배포

## 1) 환경 변수 & 토큰

google-credentials.json

```
{
  "type": "service_account",
  "project_id": "roughcode-385105",
  "private_key_id": "8e282a530657d1cc46eabc8edb0eda4227d874cd",
  "private_key": "-----BEGIN PRIVATE KEY-----\nMIIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKggggSkAgEAAoIBAQC8JktN2QGhEz/p\n0PbzPLVubx3f1oDaD\n",
  "client_email": "cody-191@roughcode-385105.iam.gserviceaccount.com",
  "client_id": "11311228861155690298",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/cody-191%40roughcode-385105.iam.gserviceaccount.com"
}
```

.env

```
CREDENTIAL_PATH=google-credentials.json;
HOST=k8a306.p.ssafy.io;
JWT_SECRET=wjdtjdenchlrhwhwdtjdenchlrhwhwdtjdenchlrhwhwdtjdenchlrhwh;
MARIADB_PASSWORD=roughcode306;
MARIADB_USER=root;
MONGODB_DATABASE=alarm;
MONGODB_PASSWORD=roughcode306;
MONGODB_USER=ssafy306;
OAUTH2_CLIENT_ID=c9bbd6ac1716fbae6b30;
OAUTH2_CLIENT_SECRET=5f41c99a3680e8142d5c93eac35248e9268f7bdf;
REDIS_PASSWORD=roughcode306;
S3_ACCESSKEY=AKIAQEZDLRRQGTQGRWFT;
S3_SECRETKEY=tzk1YKDU29lFh7VRkxuKFj4ePeXynZYQq41Wh7Z;
EMAIL_USERNAME=306roughcode@gmail.com;
EMAIL_PASSWORD=hjdsnrgruplaxssqo
```

소나 큐브 토큰

```
sqa_8fe374d4ad360a8c8e6acc2ed1ba9cdbcec34d23
```

## 2) 빌드 과정

### ▼ 자동 배포

#### EC2 접근

#### Docker 설치

##### 1. 기본 설정.

- 이 명령어는 HTTPS를 사용하여 소프트웨어를 안전하게 다운로드하고, 인증서를 관리하여 보안성을 높이며, 소프트웨어 저장소를 관리할 수 있도록 필요한 패키지를 모두 설치합니다.

```
$ sudo apt install apt-transport-https ca-certificates curl software-properties-common
```

##### 1. 자동 설치 스크립트 활용

- 리눅스 배포판 종류를 자동으로 인식하여 Docker 패키지를 설치해주는 스크립트를 제공

```
$ sudo wget -qO- https://get.docker.com/ | sh
```

##### 1. Docker 서비스 실행하기 및 부팅 시 자동 실행 설정

```
$ sudo systemctl start docker
$ sudo systemctl enable docker
```

## 1. Docker 그룹에 현재 계정 추가

```
$ sudo usermod -aG docker ${USER}
$ sudo systemctl restart docker
```

- sudo를 사용하지 않고 docker를 사용할 수 있다.
- docker 그룹은 root 권한과 동일하므로 꼭 필요한 계정만 포함
- 현재 계정에서 로그아웃한 뒤 다시 로그인

## 1. Docker 설치 확인

```
$ docker -v
```

## Jenkins 설치

```
$ mkdir jenkins-docker
$ cd jenkins-docker
$ vi Dockerfile
```

## Dockerfile

```
FROM jenkins/jenkins:latest

USER root

RUN apt-get update \
    && apt-get -y install lsb-release \
    && curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg \
    && echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/debian $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null \
    && apt-get update \
    && apt-get -y install docker-ce docker-ce-cli containerd.io

RUN usermod -aG docker jenkins

USER jenkins
```

```
$ docker build -t my-jenkins:0.1 .
$ docker run -d --name jenkins -v /var/run/docker.sock:/var/run/docker.sock -v jenkins:/var/jenkins_home -p 9090:8080 my-jenkins:0.1
$ docker exec -it jenkins bash
```

비밀번호 확인 후 9090포트로 jenkins에 들어갈 수 있다.

```
jenkins$ docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

## Docker-compose 설치

jenkins 안에서 진행

```
$ docker exec -itu 0 jenkins bash

jenkins$ curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
jenkins$ chmod +x /usr/local/bin/docker-compose
jenkins$ docker-compose --version
```

## 깃랩 플러그인 설치

[GitLab Plugin](#) 1.7.12

This plugin allows [GitLab](#) to trigger Jenkins builds and display their results in the GitLab UI.

[Report an issue with this plugin](#)

## 프로젝트에 파일 추가. README 파일과 같은 위치

### ./start.sh

```
docker-compose -f docker-compose.yml pull

COMPOSE_DOCKER_CLI_BUILD=1 DOCKER_BUILDKIT=1 docker-compose -f docker-compose.yml up --build -d

docker rmi -f $(docker images -f "dangling=true" -q) || true
```

### ./docker-compose.yml

```
version: '3.8'
services:
  db:
    image: mariadb:10.6.5
    volumes:
      - db-data:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=
      - MYSQL_DATABASE=
      - MYSQL_USER=
      - MYSQL_PASSWORD=
    ports:
      - "3306:3306"
  redis:
    image: redis:6.2.6-alpine
    ports:
      - "6379:6379"
  backend:
    build:
      context: ./도커파일치는 앱이있는 위치로 설정
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
    volumes:
      - ./앱위치로 설정:/app
    depends_on:
      - db
      - redis
    environment:
      - SPRING_DATASOURCE_URL=jdbc:mariadb://j8a605.p.ssafy.io:3306/ssafy605
      - SPRING_DATASOURCE_USERNAME=ssafy605
      - SPRING_DATASOURCE_PASSWORD=ssafy605
      - REDIS_HOST=redis

volumes:
  db-data:
```

### ./앱위치/Dockerfile



```
FROM openjdk:17-jdk-slim as builder

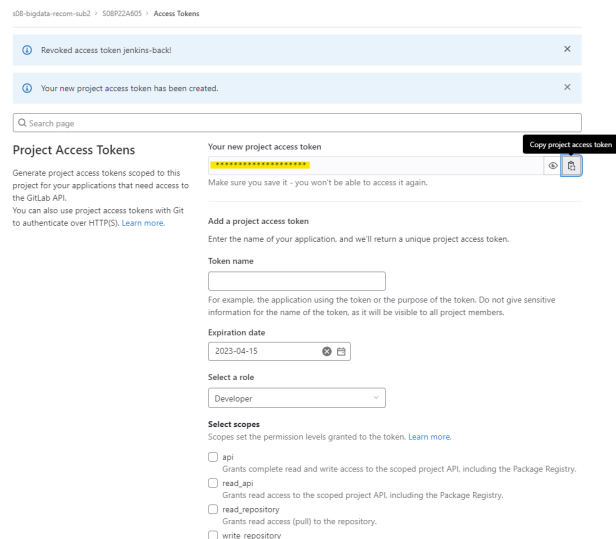
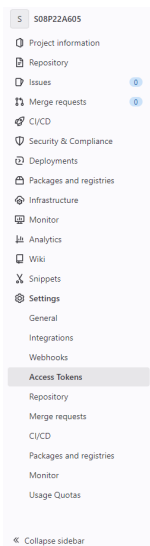
COPY gradlew .
COPY gradle gradle
COPY build.gradle .
COPY settings.gradle .
COPY src src
RUN chmod +x ./gradlew
RUN ./gradlew bootJar

FROM openjdk:17-jdk-slim
COPY --from=builder build/libs/*.jar app.jar
EXPOSE 8080

ENTRYPOINT ["java", "-Duser.timezone=Asia/Seoul", "-jar", "/app.jar"]
```

## 깃랩 젠킨스 연동

gitlab에서 토큰 복사



빨간색 부분에 붙여넣기

주소 : 토큰 @ 깃주

### Configure

- General
- 소스 코드 관리**
- 빌드 유발
- 빌드 환경
- Build Steps
- 빌드 후 조치

### 소스 코드 관리

☐ None

☒ Git

Repositories

Repository URL

Credentials

그룹

Branches to build

Branch Specifier (blank for 'any')

## Configure

- General
- 소스 코드 관리
- 빌드 유발
- 빌드 환경
- Build Steps**
- 빌드 후 조치

### Build Steps

Execute shell

Command

See the list of available environment variables

chmod +x start.sh

고급

Execute shell

Command

See the list of available environment variables

./start.sh

고급

## Webhook 설정

### Configure

- General
- 소스 코드 관리**
- 빌드 유발
- 빌드 환경
- Build Steps
- 빌드 후 조치

☐ Build after other projects are built
 ☐ Build periodically
 ☒ Build when a change is pushed to GitLab

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 (helps) for triggering a build
 

Jenkins please retry a build

고급

### Configure

- General
- 소스 코드 관리**
- 빌드 유발
- 빌드 환경
- Build Steps
- 빌드 후 조치

고급

☒ Enable [ci-skip]
 ☒ Ignore WIP Merge Requests

Labels that forces builds 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

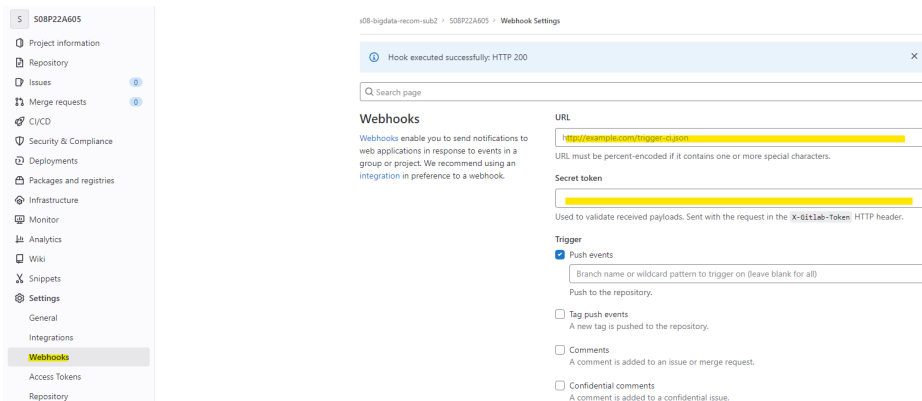
☐ Cancel pending merge request builds on update
 Allowed branches
 ☒ Allow all branches to trigger this job
 ☐ Filter branches by name
 ☐ Filter branches by regex
 ☐ Filter merge request by label

Secret token

Generate

Clear

## 깃랩



## 중요한 변수 숨기기

젠킨스 /var/젠킨스 위치 에 .env 파일로 중요 변수들을 설정해두고 .env 파일로 복사한다.

빌드 전에 젠킨스에있는 .env 파일을 .env 파일로 복사해서 도커 컴포즈가 빌드 될때 같이 넣어주면된다.

Command

See [the list of available environment variables](#)

```
cp /var/jenkins_home/.env .env
docker-compose -f docker-compose-dev.yml --env-file=.env down
docker-compose -f docker-compose-dev.yml --env-file=.env build
docker-compose -f docker-compose-dev.yml --env-file=.env up -d

chmod +x start-dev.sh
```

고급 ▾

Command

See [the list of available environment variables](#)

```
./start-dev.sh
```

1. `docker-compose -f docker-compose-dev.yml --env-file=.env down`

이 명령어는 `docker-compose-dev.yml` 파일을 사용하여 정의된 모든 서비스와 관련 리소스 (네트워크, 볼륨 등)를 중지하고 제거합니다. `--env-file=.env` 는 환경 변수를 포함하는 파일을 지정합니다. 이를 통해, 서비스 설정에 필요한 환경 변수 값을 제공할 수 있습니다.

2. `docker-compose -f docker-compose-dev.yml --env-file=.env build`

이 명령어는 `docker-compose-dev.yml` 파일에 정의된 서비스를 빌드합니다. 이 과정에서, 각 서비스에 대해 Docker 이미지가 생성됩니다. 이 때, `--env-file=.env`를 사용하여 환경 변수 파일을 지정하여 빌드 과정에 필요한 환경 변수를 제공합니다.

3. `docker-compose -f docker-compose-dev.yml --env-file=.env up -d`

이 명령어는 `docker-compose-dev.yml` 파일에 정의된 서비스를 시작하고, 이전에 빌드한 이미지를 기반으로 컨테이너를 생성하고 실행합니다. `-d` 옵션은 컨테이너를 백그라운드에서 실행하도록 지시합니다. `--env-file=.env`를 사용하여 환경 변수 파일을 지정하여 실행 중인 서비스에 필요한 환경 변수를 제공합니다.

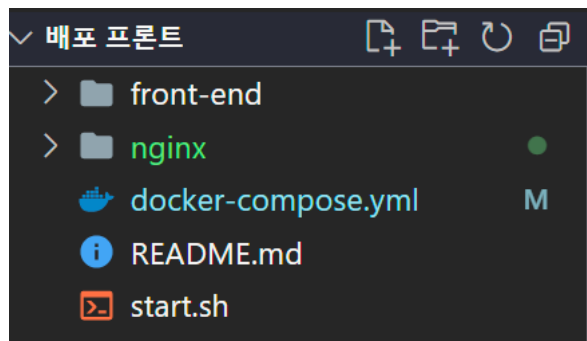
요약하면, 이 3개의 명령어는 다음과 같은 순서로 작동합니다.

1. 기존의 Docker 컨테이너와 리소스를 중지하고 제거합니다.
2. 정의된 서비스에 대한 Docker 이미지를 빌드합니다.
3. 빌드된 이미지를 기반으로 Docker 컨테이너를 생성하고 백그라운드에서 실행합니다.

## 리액트와 Nginx 같이 띄우기

<https://brick-house.tistory.com/15>

참고 사이트



폴더 구조

client

- 내 리액트 앱
- Dockerfile

nginx

- default.conf
- Dockerfile

nginx/default.conf

```
upstream 내가설정한_프론트이미지이름 {
    server 내가설정한_프론트이미지이름:3000;
}

server {
    listen 80;
    server_name rough-code.com; #도메인 연

    location / {
```

```

        proxy_pass http://내가설정한 프론트이미지이름;
    }

    location /sockjs-node {
        proxy_pass http://내가설정한 프론트이미지이름;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";
    }
}

```

```

upstream frontend {
    server frontend:3000;
}

server {
    listen 80;
    server_name rough-code.com;

    location / {
        proxy_pass http://frontend;
    }

    location /sockjs-node {
        proxy_pass http://frontend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";
    }
}

```

#### /nginx/Dockerfile

```

FROM nginx
COPY ./default.conf /etc/nginx/conf.d/default.conf

```

#### /docker-compose.yml

```

version: "3.8"

services:
  frontend:
    build:
      context: ./front-end
      dockerfile: Dockerfile
    image: frontend
    restart: always
    ports:
      - 3000:3000
    environment:
      - NEXT_PUBLIC_API_URL=${NEXT_PUBLIC_API_URL}
    networks:
      - app-network

  nginx:
    restart: always
    container_name: nginx
    build:
      context: ./nginx
      dockerfile: Dockerfile
    ports:
      - "80:80"
    networks:
      - app-network

networks:
  app-network:
    driver: bridge

```

# HTTPS 설정

## 1단계 - Certbot으로 SSL 인증서 발급 받기

docker-compose.yml

```
#nginx 이미지
build:
  context: ./nginx
  dockerfile: Dockerfile

# certbot 관련 볼륨 추가
# 아래의 certbot 볼륨과 경로가 똑같아야 한다.
volumes:
  - ./data/certbot/conf:/etc/letsencrypt
  - ./data/certbot/data:/var/www/certbot

# 443 포트 추가
ports:
  - "80:80"
  - "443:443"

#certbot 이미지 생성
certbot:
  image: certbot/certbot:latest
  command: certonly --webroot --webroot-path=/var/www/certbot --email 내이메일 --agree-tos --no-eff-email -d 도메인 -d www.도메인
  volumes:
    - ./data/certbot/conf:/etc/letsencrypt
    - ./data/certbot/logs:/var/log/letsencrypt
    - ./data/certbot/data:/var/www/certbot
```

./nginx/default.conf

```
server {
    listen 80;
    server_name rough-code.com www.rough-code.com;

    #인증서 발급용 통신 url 작성
    location ~ /.well-known/acme-challenge {
        allow all;
        root /var/www/certbot;
    }

    location / {
        proxy_pass http://frontend;
    }

    location /sockjs-node {
        proxy_pass http://frontend;
        proxy_http_version 1.1;
    }
}
```

전체 파일

docker-compose.yml

```
version: "3.8"

services:
```

```

frontend:
  build:
    context: ./front-end
    dockerfile: Dockerfile
  image: frontend
  restart: always
  ports:
    - 3000:3000
  environment:
    - NEXT_PUBLIC_API_URL=${NEXT_PUBLIC_API_URL}
  networks:
    - app-network

nginx:
  restart: always
  container_name: nginx
  build:
    context: ./nginx
    dockerfile: Dockerfile
  # 인증서 볼륨 설정
  volumes:
    # - ./nginx:/etc/nginx/conf.d
    - ./data/certbot/conf:/etc/letsencrypt
    - ./data/certbot/data:/var/www/certbot

  ports:
    - "80:80"
    # 443 포트 확장
    - "443:443"

  command: '/bin/sh -c \'while :; do sleep 6h & wait ${!}; nginx -s reload; done & nginx -g "daemon off;"\'

  networks:
    - app-network

#certbot 이미지 생성.
certbot:
  image: certbot/certbot:latest
  container_name: certbot
  command: certonly --webroot --webroot-path=/var/www/certbot --email dongsu712@naver.com --agree-tos --no-eff-email -d rough
  volumes:
    - ./data/certbot/conf:/etc/letsencrypt
    - ./data/certbot/data:/var/www/certbot
    - ./data/certbot/logs:/var/log/letsencrypt

  networks:
    app-network:
      driver: bridge

```

certbot 명령어 설명  
 command: certonly --webroot --webroot-path=/var/www/certbot --email dongsu712@naver.com --agree-tos --no-eff-email -d rough-code.com

이 명령어는 Let's Encrypt에서 제공하는 certbot 도구를 사용하여 SSL/TLS 인증서를 발급받는 데 사용됩니다. 각 옵션들은 다음과 같은 의미를 가지고 있습니다.

certonly: 인증서를 발급만 받고, 웹 서버 설정을 변경하지 않도록 지시합니다. 이렇게 하면 사용자가 웹 서버 설정을 수동으로 변경할 수 있습니다.

--webroot: 인증서 발급을 위한 웹 루트 방식을 사용하도록 지시합니다. 이 방식은 웹 서버의 루트 디렉토리에 인증 파일을 저장하여 도메인 소유권을 확인합니다.

--webroot-path=/var/www/certbot: 인증 파일을 저장할 웹 루트 디렉토리의 경로를 지정합니다.

--email dongsu712@naver.com: 인증서 만료 알림 및 복구 관련 이메일을 받을 주소를 지정합니다.

--agree-tos: Let's Encrypt의 서비스 약관에 동의하도록 지시합니다.

--no-eff-email: Electronic Frontier Foundation의 이메일 목록에 등록하지 않도록 지시합니다. 이것은 추가 정보 및 뉴스레터를 받지 않겠다는 의미입니다.

-d rough-code.com: 인증서를 발급받을 도메인 이름을 지정합니다.

이 명령어를 실행하면, Let's Encrypt는 지정한 도메인에 대해 SSL/TLS 인증서를 발급하고, 발급된 인증서와 개인 키를 저장할 디렉토리를 생성합니다. 이후 사용자는

## default.conf

```

# 프론트 서버와 연결
upstream frontend {
    server frontend:3000;
}

server {
    listen 80;

```

```

server_name rough-code.com;

# return 301 https://$host$request_uri;

# 인증서 첫 발급 세팅
location ~ /.well-known/acme-challenge {
    allow all;
    root /var/www/certbot;
}

# nginx 세팅
location / {
    proxy_pass http://frontend;
}

# nginx 세팅
location /sockjs-node {
    proxy_pass http://frontend;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "Upgrade";
}

}

# SSL 설정을 위한 새 server 블록 추가
# server {
#     listen 443 ssl;
#     server_name rough-code.com;

#     ssl_certificate /etc/letsencrypt/live/rough-code.com/fullchain.pem;
#     ssl_certificate_key /etc/letsencrypt/live/rough-code.com/privkey.pem;
#     # ssl_certificate /etc/nginx/ssl/live/rough-code.com/fullchain.pem;
#     # ssl_certificate_key /etc/nginx/ssl/live/rough-code.com/privkey.pem;

# # # nginx 세팅
# #     location / {
# #         proxy_pass http://frontend;
# #     }

# # # nginx 세팅
# #     location /sockjs-node {
# #         proxy_pass http://frontend;
# #         proxy_http_version 1.1;
# #         proxy_set_header Upgrade $http_upgrade;
# #         proxy_set_header Connection "Upgrade";
# #     }
# # }

```

## 2단계 - 인증서 발급 후 인증서 재발급 코드로 변경하기

docker-compose.yml

certbot 명령어 부분을 바꿔준다.

```

version: "3.8"

services:
  frontend:
    build:
      # 어디에서 도커를 시작할 것인가
      context: ./front-end
      dockerfile: Dockerfile
    image: frontend
    restart: always
    ports:
      - 3000:3000
    environment:
      - NEXT_PUBLIC_API_URL=${NEXT_PUBLIC_API_URL}
    networks:
      - app-network

  nginx:
    restart: always
    container_name: nginx

```



```

build:
  context: ./nginx
  dockerfile: Dockerfile

volumes:
  - ./data/certbot/conf:/etc/letsencrypt
  - ./data/certbot/data:/var/www/certbot

ports:
  - "80:80"
  - "443:443"
command: '/bin/sh -c \'while ;; do sleep 6h & wait $$(!); nginx -s reload; done & nginx -g "daemon off;"\'

networks:
  - app-network

certbot:
  image: certbot/certbot:latest
  container_name: certbot

# 발급후에 커맨드 아래 재발급용 커맨드로 바꿔준다.
# command: certonly --webroot --webroot-path=/var/www/certbot --email dongsu712@naver.com --agree-tos --no-eff-email -d rou
entrypoint: "/bin/sh -c 'trap exit TERM; while ;; do certbot renew; sleep 12h & wait $$(!); done;'"
volumes:
  - ./data/certbot/conf:/etc/letsencrypt
  - ./data/certbot/data:/var/www/certbot
  - ./data/certbot/logs:/var/log/letsencrypt

networks:
  app-network:
    driver: bridge

```

#### certbot entrypoint 명령어 설명

```
/bin/sh -c 'trap exit TERM; while ;; do certbot renew; sleep 12h & wait $$(!); done;'
```

이 명령어는 인증서를 자동으로 갱신하는 데 사용되는 쉘 스크립트입니다. 다음과 같은 작업을 수행합니다:

```
/bin/sh를 사용하여 쉘 스크립트를 실행합니다.
trap exit TERM를 통해 TERM 시그널이 발생하면 스크립트가 종료되도록 설정합니다.
무한 루프(while ;;)를 통해 다음 작업을 반복 수행합니다:
certbot renew: 인증서를 갱신하려고 시도합니다. 인증서가 만료되기 직전일 경우에만 갱신됩니다.
sleep 12h & wait $$(!): 12시간 동안 대기한 후에 다시 인증서 갱신을 시도합니다.
```

#### default.conf

```

upstream frontend {
    server frontend:3000;
}

server {
    listen 80;
    server_name rough-code.com;

    # 인증서 발급용 코드 제거.
    # location ~ /.well-known/acme-challenge {
    #     allow all;
    #     root /var/www/certbot;
    # }

    # 80 포트로 오는 모든 요청을 https로 보내는 코드.
    location / { # https로 요청
        return 301 https://$host$request_uri;
    }

    # nginx 세팅 코드 443 포트로 이동
    # nginx 세팅
    # location / {
    #     proxy_pass http://frontend;
    # }

    # nginx 세팅
    # location /sockjs-node {
    #     proxy_pass http://frontend;
    # }

```

```

# proxy_http_version 1.1;
# proxy_set_header Upgrade $http_upgrade;
# proxy_set_header Connection "Upgrade";
# }

}

# SSL 설정을 위한 새 server 블록 추가
server {
    listen 443 ssl;
    server_name rough-code.com;

    #docker-compose에서 볼륨으로 설정해준 위치와 같도록 설정해준다.
    ssl_certificate /etc/letsencrypt/live/rough-code.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/rough-code.com/privkey.pem;

    # # nginx 세팅
    location / {
        proxy_pass http://frontend;
    }

    # # nginx 세팅
    location /sockjs-node {
        proxy_pass http://frontend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";
    }
}

```

## 백엔드 통신 HTTPS 설정

default.conf

443 포에 추가

```

location /api {
    # 백엔드 서버에 맞춰 수정
    proxy_pass http://k8a306.p.ssafy.io:8080;

    proxy_set_header Host $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
}

```

프론트엔드 .env 파일 수정.

NEXT\_PUBLIC\_API\_URL 이 <https://rough-code.com/api> 로 가계함

```

NEXT_PUBLIC_API_URL=https://rough-code.com/api
#NEXT_PUBLIC_API_URL=http://k8a306.p.ssafy.io:8080/api/v1

```

## ▼ 무중단 배포

### ▼ 스프링 무중단 배포

## 우분투 로컬 세팅

### 우분투에 nginx 컨테이너 직접 설치

(초기 설정만 필요하고 계속 Up 상태로 두는 이미지가기 때문에 컴포즈가 아닌 로컬에 직접 설치)

우분투 로컬에 폴더를 하나 만들어서 해당 폴더 안에서 아래 파일을 만들고 명령어를 실행한다.

#### ./back-nginx-Dockerfile

```
FROM nginx

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

COPY ./conf.d/app.conf /etc/nginx/conf.d/app.conf
COPY ./conf.d/nginx.conf /etc/nginx/nginx.conf

VOLUME ["/data", "/etc/nginx", "/var/log/nginx"]

WORKDIR /etc/nginx

CMD ["nginx"]

EXPOSE 8080
```

#### ./conf.d/app.conf

```
server {
    listen 8080;
    listen [::]:8080;

    server_name "";

    access_log off;

    location / {
        proxy_pass http://docker-app;
        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 http;
        proxy_max_temp_file_size 0;

        proxy_connect_timeout 150;
        proxy_send_timeout 100;
        proxy_read_timeout 100;

        proxy_buffer_size 8k;
        proxy_buffers 4 32k;
        proxy_busy_buffers_size 64k;
        proxy_temp_file_write_size 64k;
    }
}
```

#### ./conf.d/nginx.conf

```
daemon off;
user www-data;
worker_processes 2;

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

events {
    worker_connections 1024;
    use epoll;
    accept_mutex off;
}

http {
    include /etc/nginx/mime.types;
```

```

proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

default_type application/octet-stream;

upstream docker-app {
    least_conn;
    ## 가동되는 백엔드 컨테이너와 매칭 시키는 부분.
    server k8a306.p.ssafy.io:8082 weight=10 max_fails=3 fail_timeout=30s;
    server k8a306.p.ssafy.io:8083 weight=10 max_fails=3 fail_timeout=30s;
}

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

sendfile on;
#tcp_nopush

keepalive_timeout 65;

client_max_body_size 300m;
client_body_buffer_size 128k;

gzip on;
gzip_http_version 1.0;
gzip_comp_level 6;
gzip_min_length 0;
gzip_buffers 16 8k;
gzip_proxied any;
gzip_types text/plain text/css text/xml text/javascript application/xml application/xml+rss application/javascript application/javascript+rss;
gzip_disable "MSIE [1-6]\.";
gzip_vary on;

#리눅스환경에서 취급하는 호스팅하는 웹서버 경로
include /etc/nginx/conf.d/*.conf;
}

```

## Nginx 이미지 만들고 실행

우분투 로컬 해당 폴더 위치에서

```

docker build -t back-nginx:0.1 -f back-nginx-Dockerfile .

docker run -d --name back-nginx -p 8080:8080 back-nginx:0.1

```

## 스프링 프로젝트 세팅

### ./docker-compose.blue.yml

이미지 이름과 컨테이너 네임을 설정해주고, 포트는 빈포트:8080 으로 설정

나머지는 기존 컴포즈 내용을 가져온다.

```

version: "3.8"

services:
  app:
    image: app:0.1
    container_name: app_blue
    environment:
      - "spring_profiles_active=blue"
      - S3_ACCESSKEY=${S3_ACCESSKEY}
      - S3_SECRETKEY=${S3_SECRETKEY}
      - HOST=${HOST}
      - JWT_SECRET=${JWT_SECRET}

```

```

- MARIADB_PASSWORD=${MARIADB_PASSWORD}
- MARIADB_USER=${MARIADB_USER}
- MONGODB_DATABASE=${MONGODB_DATABASE}
- MONGODB_PASSWORD=${MONGODB_PASSWORD}
- MONGODB_USER=${MONGODB_USER}
- OAUTH2_CLIENT_ID=${OAUTH2_CLIENT_ID}
- OAUTH2_CLIENT_SECRET=${OAUTH2_CLIENT_SECRET}
- REDIS_PASSWORD=${REDIS_PASSWORD}
- CREDENTIAL_PATH=${CREDENTIAL_PATH}
- EMAIL_USERNAME=${EMAIL_USERNAME}
- EMAIL_PASSWORD=${EMAIL_PASSWORD}

ports:
- "8082:8080"

volumes:
- ./back-end:/app

external_links:
- dev-back_db_1:db
- dev-back_mongodb_1:mongodb
- dev-back_redis_1:redis

```

## docker-compose.green.yml

```

version: "3.8"

services:
  app:
    image: app:0.2
    container_name: app_green
    environment:
      - "spring_profiles_active=blue"
      - S3_ACCESSKEY=${S3_ACCESSKEY}
      - S3_SECRETKEY=${S3_SECRETKEY}
      - HOST=${HOST}
      - JWT_SECRET=${JWT_SECRET}
      - MARIADB_PASSWORD=${MARIADB_PASSWORD}
      - MARIADB_USER=${MARIADB_USER}
      - MONGODB_DATABASE=${MONGODB_DATABASE}
      - MONGODB_PASSWORD=${MONGODB_PASSWORD}
      - MONGODB_USER=${MONGODB_USER}
      - OAUTH2_CLIENT_ID=${OAUTH2_CLIENT_ID}
      - OAUTH2_CLIENT_SECRET=${OAUTH2_CLIENT_SECRET}
      - REDIS_PASSWORD=${REDIS_PASSWORD}
      - CREDENTIAL_PATH=${CREDENTIAL_PATH}
      - EMAIL_USERNAME=${EMAIL_USERNAME}
      - EMAIL_PASSWORD=${EMAIL_PASSWORD}
    ports:
      - "8083:8080"
    volumes:
      - ./back-end:/app

    external_links:
      - dev-back_db_1:db
      - dev-back_mongodb_1:mongodb
      - dev-back_redis_1:redis

```

## 프로젝트 최상단에 빌드용 shell 스크립트 작성

### ./deploy.sh

```

#!/bin/bash

function create_docker_image_blue(){

  echo "> blue docker image 만들기"

  # 해당 프로젝트 위치로 이동 한 후 빌드 해주고
  cd back-end/roughcode

  ./gradlew clean build

  docker build -t app:0.1 .

  # 다시 최상단 이동
  cd ..
  cd ..

```

```

}

function create_docker_image_green(){

    echo "> green docker image 만들기"

    cd back-end/roughcode

    ./gradlew clean build

    docker build -t app:0.2 .

    cd ..
    cd ..
}

function execute_blue(){
    docker ps -q --filter "name=app_blue" || grep -q . && docker stop app_blue && docker rm app_blue || true

    sleep 10

    docker-compose -p app-blue -f docker-compose.blue.yml up -d

    sleep 10

    echo "GREEN:8083 종료"
    docker-compose -p app-green -f docker-compose.green.yml down

    #dangling=true : 불필요한 이미지 지우기
    docker rmi -f $(docker images -f "dangling=true" -q) || true
}

function execute_green(){
    docker ps -q --filter "name=app_green" || grep -q . && docker stop app_green && docker rm app_green || true

    echo "GREEN:8083 실행"
    docker-compose -p app-green -f docker-compose.green.yml up -d

    sleep 10

    echo "BLUE:8082 종료"
    docker-compose -p app-blue -f docker-compose.blue.yml down

    #dangling=true : 불필요한 이미지 지우기
    docker rmi -f $(docker images -f "dangling=true" -q) || true
}

# 현재 사용중인 어플리케이션 확인
# 8083포트의 값이 없으면 8082포트 사용 중
# shellcheck disable=SC2046
RUNNING_GREEN=$(docker ps -aqf "name=app_green")
RUNNING_BLUE=$(docker ps -aqf "name=app_blue")

echo ${RUNNING_GREEN}
echo ${RUNNING_BLUE}

# Blue or Green
if [ -z ${RUNNING_GREEN} ]
then
    # 초기 실행 : BLUE도 실행중이지 않을 경우
    if [ -z ${RUNNING_BLUE} ]
    then
        echo "구동 앱 없음 => BLUE 실행"

        create_docker_image_blue

        sleep 10

        docker-compose -p app-blue -f docker-compose.blue.yml up -d

    else
        # 8083포트로 어플리케이션 구동
        echo "BLUE:8082 실행 중"

        create_docker_image_green

        execute_green
    fi
else
    # 8082포트로 어플리케이션 구동
    echo "GREEN:8083 실행 중"

    echo "BLUE:8082 실행"

    create_docker_image_blue

```

```
execute_blue

fi

# 새로운 어플리케이션 구동 후 현재 어플리케이션 종료
#kill -15 ${RUNNING_PORT_PID}
```

## Jenkins 세팅

해당 프로젝트 → 구성 → BuildSteps → Execute shell

도커 이미지를 만들기 위해 빌드를 하고

그 빌드 파일로 Blue, Green 컨테이너를 생성하는 명령과정

```
cd back-end/roughcode
chmod +x ./gradlew
./gradlew bootJar
cd ..
cd ..
chmod +x deploy.sh
./deploy.sh
```

### ▼ Next.js 무중단 배포

프론트는 기존 certbot과 연결된 설정이 있어. 도커 컴포즈로 만들어진 nginx 계속 사용

프론트 프로젝트 세팅

#### ./docker-compose.yml

기존 프론트엔드 이미지와 네트워크 세팅 부분 전부 제거.

```
version: "3.8"

services:
  # frontend:
  #   build:
  #     # 어디에서 도커를 시작할 것인가
  #     context: ./front-end
  #     dockerfile: Dockerfile
  #   image: frontend
  #   restart: always

  # #####
  #   volumes:
  #     - /app/node_modules #도커 /app/node_modules는 맵핑을 따로 안해주겠다.
  #     - ./:/app # 로컬에 있는 모든 파일을 맵핑

  # #####
  #   ports:
  #     # - 3000:3000
  #     - 3001:3001
  #   #####
  #   stdin_open: true

  #   environment:
  #     - NEXT_PUBLIC_API_URL=${NEXT_PUBLIC_API_URL}
  #   networks:
  #     - app-network
```

```

nginx:
  restart: always
  container_name: nginx
  build:
    context: ../nginx
    # context: ../front-end
    dockerfile: Dockerfile
  #
  volumes:
    # - ../nginx:/etc/nginx/conf.d
    - ../data/certbot/conf:/etc/letsencrypt
    - ../data/certbot/data:/var/www/certbot

  ports:
    - "80:80"
    - "443:443"
  #
  command: '/bin/sh -c \'while :; do sleep 6h & wait ${!}; nginx -s reload; done & nginx -g "daemon off;\'\'\'

  # networks:
  #   - app-network

certbot:
  image: certbot/certbot:latest
  container_name: certbot

  # 발급후에 커맨드 아래 재발급용 커맨드로 바꿔준다.
  # command: certonly --webroot --webroot-path=/var/www/certbot --email dongsu712@naver.com --agree-tos --no-eff-email -d
  entrypoint: "/bin/sh -c 'trap exit TERM; while :; do certbot renew; sleep 12h & wait ${!}; done;'"
  volumes:
    - ../data/certbot/conf:/etc/letsencrypt
    - ../data/certbot/data:/var/www/certbot
    - ../data/certbot/logs:/var/log/letsencrypt
  # networks:
  #   app-network:
  #     driver: bridge

```

## ./nginx/default.conf

```

## 추가된 부분. 3002(blue) 3003(green) 컨테이너와 연결.
upstream docker-app {
    least_conn;
    server k8a306.p.ssafy.io:3002 weight=10 max_fails=3 fail_timeout=30s;
    server k8a306.p.ssafy.io:3003 weight=10 max_fails=3 fail_timeout=30s;
}

server {
    listen 80;
    server_name rough-code.com;

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

server {
    listen 443 ssl;
    server_name rough-code.com;

    ssl_certificate /etc/letsencrypt/live/rough-code.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/rough-code.com/privkey.pem;

## 추가된 부분
    location / {
        proxy_pass http://docker-app;
        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 http;
        proxy_max_temp_file_size 0;

        proxy_connect_timeout 150;
        proxy_send_timeout 100;
        proxy_read_timeout 100;

        proxy_buffer_size 8k;
        proxy_buffers 4 32k;
        proxy_busy_buffers_size 64k;

```



```

        proxy_temp_file_write_size 64k;
    }

    location /sockjs-node {
        ## 바뀐 앱이름에 맞추어 수정한 부분
        proxy_pass http://docker-app;
        # proxy_pass http://frontend;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";
    }

    location /api {
        proxy_pass http://k8a306.p.ssafy.io:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}

```

### ./nginx/Dockerfile

```

FROM nginx

EXPOSE 3000

COPY ./default.conf /etc/nginx/conf.d/default.conf

```

## Blue, Green 컴포즈 파일 추가

### ./docker-compose.blue.yml

```

version: "3.8"

services:
  front:
    image: front:0.1
    container_name: front_blue
    environment:
      - NEXT_PUBLIC_API_URL=${NEXT_PUBLIC_API_URL}
    ports:
      - "3002:3000"

```

### ./docker-compose.green.yml

```

version: "3.8"

services:
  front:
    image: front:0.2
    container_name: front_green
    environment:
      - NEXT_PUBLIC_API_URL=${NEXT_PUBLIC_API_URL}
    ports:
      - "3003:3000"

```

## 셸 실행 용 deploy.sh 추가

./deploy.sh

```
#!/bin/bash

function create_docker_image_blue(){

    echo "> blue front docker image 만들기"
    # 프론트 프로젝트로 이동
    cd front-end
    # 프로젝트 빌드
    pnpm run build
    # 도커 이미지 생성
    docker build -t front:0.1 .
    # 최상단으로 이동
    cd ..
}

function create_docker_image_green(){

    echo "> green front docker image 만들기"

    cd front-end

    pnpm run build

    docker build -t front:0.2 .

    cd ..
}

function execute_blue(){
    docker ps -q --filter "name=front_blue" || grep -q . && docker stop front_blue && docker rm front_blue || true

    sleep 10

    docker-compose -p front-blue -f docker-compose.blue.yml up -d

    sleep 10

    echo "GREEN:3003 종료"
    docker-compose -p front-green -f docker-compose.green.yml down

    #dangling=true : 불필요한 이미지 지우기
    docker rmi -f $(docker images -f "dangling=true" -q) || true
}

function execute_green(){
    docker ps -q --filter "name=front_green" || grep -q . && docker stop front_green && docker rm front_green || true

    echo "GREEN:3003 실행"
    docker-compose -p front-green -f docker-compose.green.yml up -d

    sleep 10

    echo "BLUE:3002 종료"
    docker-compose -p front-blue -f docker-compose.blue.yml down

    #dangling=true : 불필요한 이미지 지우기
    docker rmi -f $(docker images -f "dangling=true" -q) || true
}

# 현재 사용중인 어플리케이션 확인
# 3003포트의 값이 없으면 3002포트 사용 중
# shellcheck disable=SC2046
RUNNING_GREEN=$(docker ps -aqf "name=front_green")
RUNNING_BLUE=$(docker ps -aqf "name=front_blue")

echo ${RUNNING_GREEN}
echo ${RUNNING_BLUE}

# Blue or Green
if [ -z ${RUNNING_GREEN} ]
then
    # 초기 실행 : BLUE도 실행중이지 않을 경우
    if [ -z ${RUNNING_BLUE} ]
    then
        echo "구동 앱 없음 => BLUE 실행"

        create_docker_image_blue

        sleep 10
    fi
fi
```

```

docker-compose -p front-blue -f docker-compose.blue.yml up -d

else
# 30003포트로 어플리케이션 구동
echo "BLUE:3002 실행 중"

create_docker_image_green

execute_green

fi

else
# 3002포트로 어플리케이션 구동
echo "GREEN:3003 실행 중"

echo "BLUE:3002 실행"

create_docker_image_blue

execute_blue

fi

# 새로운 어플리케이션 구동 후 현재 어플리케이션 종료
#kill -15 ${RUNNING_PORT_PID}

```

## 젠킨스 세팅

해당 프로젝트 → 구성 → Build Steps → Execute shell

```

# 프론트 프로젝트로 이동
cd front-end

# pnpm 설치
npm install -g pnpm@5

# 패키지 설치
pnpm install

# 빌드
pnpm run build

# 최상단 이동
cd ..

# deploy.sh에 권한 부여
chmod +x deploy.sh

# deploy.sh 실행
./deploy.sh

```

### ▼ SonarQube

#### ▼ 소나 큐브 이미지 생성

### 도커 컴포즈로 소나큐브, postgresql 이미지 생성하기.

프로젝트 최상단에 docker-compose 파일작성

docker-compose.sonar.yml 로 작성하였음

소나큐브와 postgresql 이미지 생성 후 네트워크로 연결해준다.

```

version: "3.8" # 사용하는 도커 버전에 맞춰 작성

```

```

services:
  sonarqube:
    image: sonarqube:lts
    depends_on:
      - sonardb # 아래 작성한 postgresql 이미지 이름과 맞춘다
    container_name: sonarqube
    ports:
      - "8000:9000" # 기본 9000 포트. ec2 서버 9000포트 접속이 안되어 8000 포트로 연결
    ulimits:
      nofile:
        soft: "262144"
        hard: "262144"
    networks:
      - sonarnet
    environment:
      - SONAR_JDBC_URL=jdbc:postgresql://sonardb:5432/sonar
      - SONAR_JDBC_USERNAME=sonar
      - SONAR_JDBC_PASSWORD=sonar
    volumes:
      - sonarqube_conf:/opt/sonarqube/conf
      - sonarqube_data:/opt/sonarqube/data
      - sonarqube_extensions:/opt/sonarqube/extensions
      - sonarqube_logs:/opt/sonarqube/logs

  sonardb:
    image: postgres:latest
    container_name: postgres_sonar
    ports:
      - "5432:5432"
    networks:
      - sonarnet
    environment:
      - POSTGRES_USER=sonar
      - POSTGRES_PASSWORD=sonar
    volumes:
      - postgresql:/var/lib/postgresql
      - postgresql_data:/var/lib/postgresql/data

networks:
  sonarnet:
    driver: bridge

volumes:
  sonarqube_conf:
  sonarqube_data:
  sonarqube_extensions:
  sonarqube_logs:
  postgresql:
  postgresql_data:

```

## 작성한 도커 컴포즈 실행

```
docker-compose -f docker-compose-sonar.yml up -d
```

### Command

See [the list of available environment variables](#)

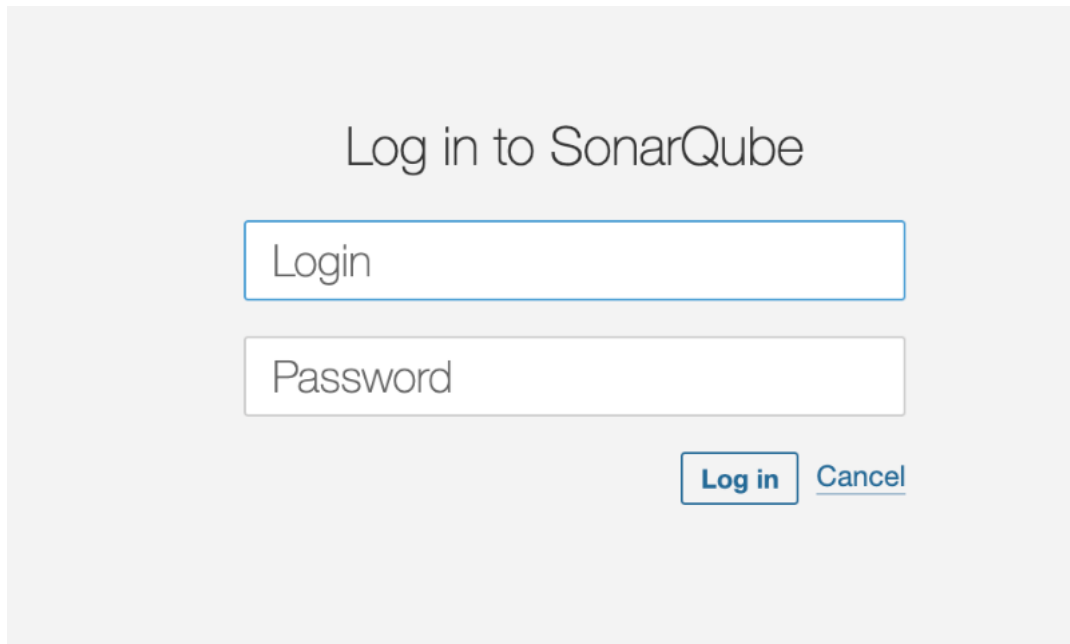
```
./start-dev.sh
```

```
# docker-compose -f docker-compose-sonar.yml stop
# docker-compose -f docker-compose-sonar.yml build
```

```
docker cp dev-back_backend_1:/app.jar /var/jenkins_home/workspace/dev-back/app/build
docker-compose -f docker-compose-sonar.yml up -d
#docker start dev-back_backend_1
```

jenkins 빌드 스텝에서 실행 하였음.

http://[EC2\_IP 또는 EC2도메인주소]:[생성한 포트번호] 로 접속하면 소나큐브 초기화면이 뜬다.

The image shows the SonarQube login interface. At the top, it says "Log in to SonarQube". Below this are two input fields: "Login" and "Password". At the bottom right, there are two buttons: "Log in" and "Cancel".

초기 로그인 정보는

admin / admin 이다

```
ERROR: [1] bootstrap checks failed. You must address the points described in the following [1] lines before starting Elasticsearch.  
bootstrap check failure [1] of [1]: max virtual memory areas vm.max_map_count [65530] is too low, increase to at least [262144]  
ERROR: Elasticsearch did not exit normally - check the logs at /opt/sonarqube/logs/sonarqube.log
```

오류해결하기

SonarQube의 Elasticsearch 구성 중 하나인 'bootstrap check'에서 발생한 것으로

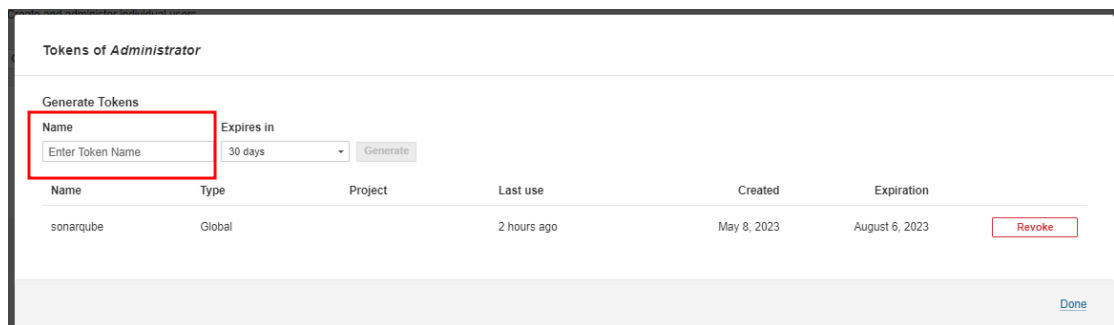
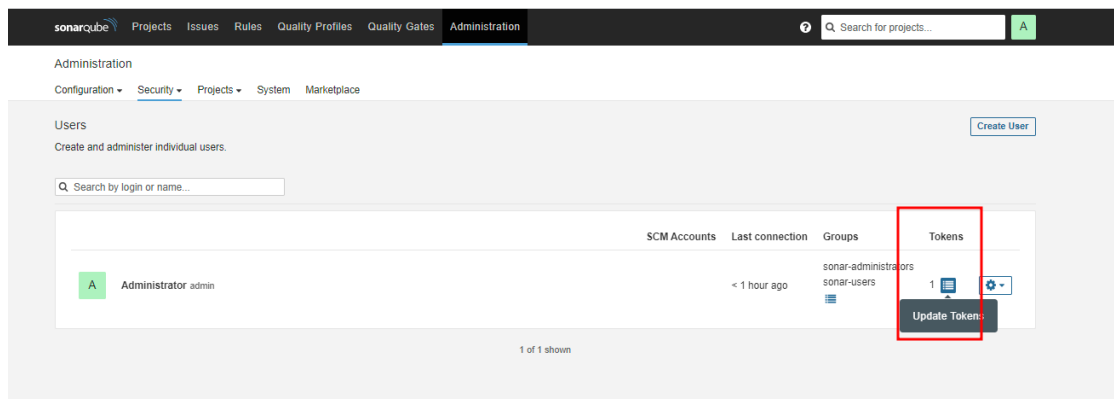
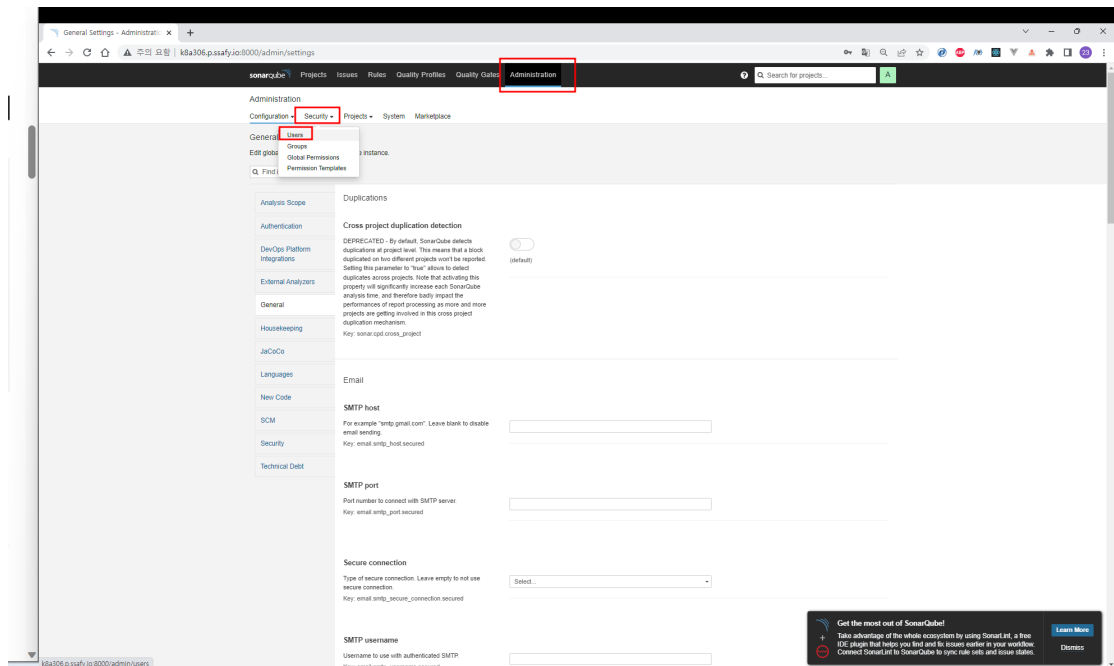
max virtual memory areas vm.max\_map\_count 가 65530으로 너무낮으니 최소 262144로 증가시키라고 요구하는 오류.

```
sudo sysctl -w vm.max_map_count=262144
```

명령어를 ec2 우분투 환경에 입력하여 최소 메모리를 증가시킨다.

## ▼ 소나 큐브 - 젠킨스 연결

### 소나큐브 토큰 생성



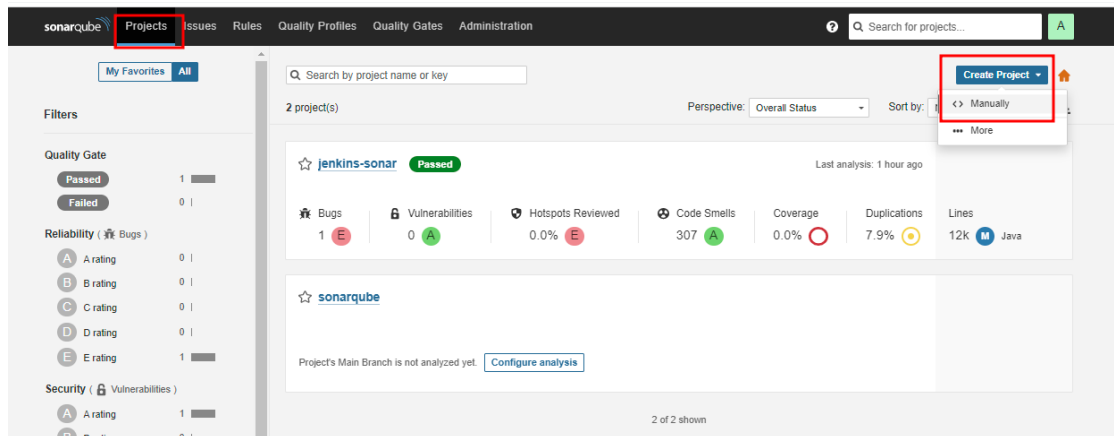
토큰 이름 작성하고 토큰 생성하기.

토큰 이름 아무거나 작성해도됨.

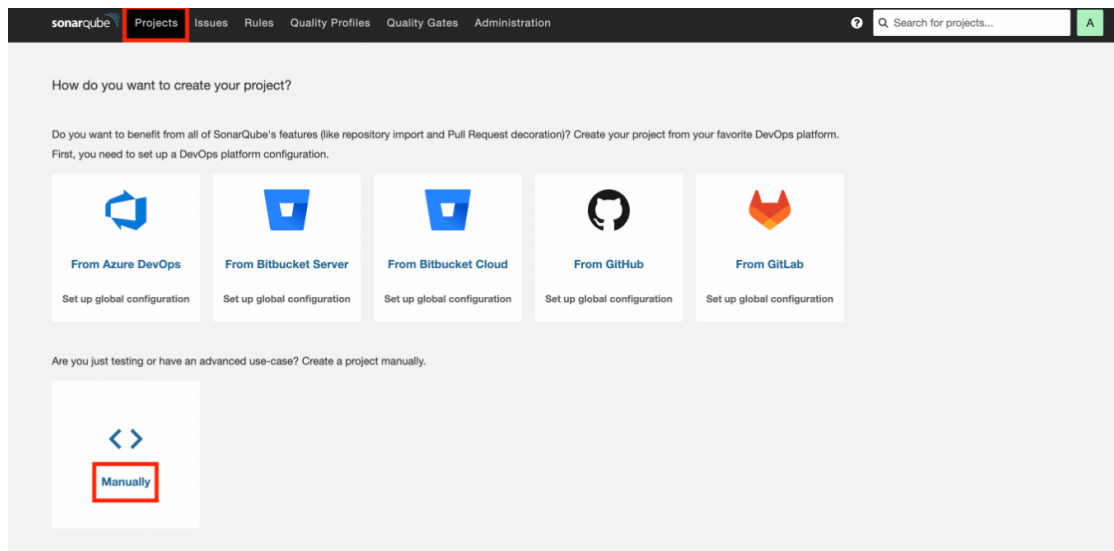
다시 확인 불가능 하니 꼭 저장.

## 소나큐브 프로젝트 생성하기

프로젝트 → Create Project → Manually



이미 만들어진 프로젝트가 있으면 위 화면  
아니면 아래 화면



원하는 프로젝트이름으로 작성  
프로젝트 키는 똑같이 설정 된다.  
프로젝트 메인 브랜치도 작성해준다.

## Create a project

All fields marked with \* are required

**Project display name \***

 ✓  
 Up to 255 characters. Some scanners might override the value you provide.
 

**Project key \***

 ✓  
 The project key is a unique identifier for your project. It may contain up to 400 characters. Allowed characters are alphanumeric, '-' (dash), '\_' (underscore), '.' (period) and ':' (colon), with at least one non-digit.
 

**Main branch name \***

 ✓  
 The name of your project's default branch [Learn More](#)

Set Up

## 젠킨스 설정

### 소나 큐브 토큰 등록

젠킨스 관리 → Security → Manage Credentials

Jenkins 신규버전(2.403)을 [여기서](#) 받을 수 있습니다.(변경사항). 또는 자동 +

Building on the built-in node can be a security issue. You should set up distributed builds. See [the documentation](#). Set up agent Set up cloud

### System Configuration











- System**  
환경변수 및 경로 정보들을 설정합니다.
- Tools**  
Configure tools, their locations and automatic installers.
- 19 Plugins**  
Jenkins의 기능을 확장하기 위한 플러그인을 추가, 제거, 사용, 미사용으로 설정할 수 있습니다.
- Nodes and Clouds**  
Add, remove, control and monitor various nodes that Jenkins runs on.
- Managed files**  
e.g. settings.xml for maven, central managed scripts, custom files, ...

### Security


- Security**  
Secure Jenkins; define who is allowed to access/use the system.
- Manage Credentials**  
Configure credentials
- Configure Credential Providers**  
Configure the credential providers and types
- Users**  
Create/delete/modify users that have access to this Jenkins.



## Credentials

T	P	Store ↓	Domain
		System	(global)
		System	(global)
		System	(global)
		System	(global)
		System	(global)

## 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
----	------	------	-------------

## New credentials

Kind

Secret text

Scope ?

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

Secret

소나큐브 시크릿 키 작성

ID ?

키 이름 설정

Description ?

Create

## 젠킨스 플러그인 설치

Jenkins 관리 → Plugins → Available plugins → SonarQube Scanner 검색 후 설치.

[SonarQube Scanner for Jenkins](#) 2.15

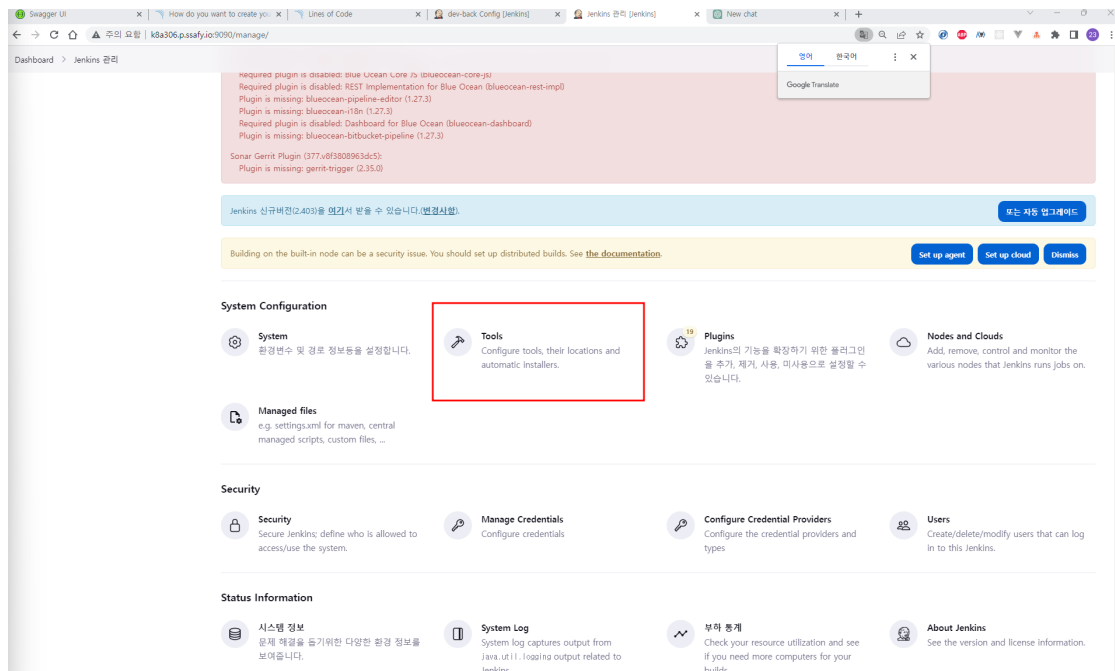
This plugin allows an easy integration of [SonarQube](#), the open source platform for Continuous Inspection of code quality.  
[Report an issue with this plugin](#)

## 플러그인 세팅

System Configuration → Tools

→ SonarQube Scanner installations

플러그인이 정상적으로 설치되어있다면 이 위치에 존재한다.



#### SonarQube Scanner installations

SonarQube Scanner installations ✎ Edited

Add SonarQube Scanner

SonarQube Scanner

Name

sonarqube

☒ Install automatically ?

Version

SonarQube Scanner 3.3.0.1492

Add Installer

Add SonarQube Scanner

**System Configuration → System**

**→ SonarQube servers**

## SonarQube servers

If checked, job administrators will be able to inject a SonarQube server configuration as environment variables in the build.

☐ Environment variables Enable injection of SonarQube server configuration as build environment variables

SonarQube installations

List of SonarQube installations

Name

SonarServer

소나 서버 이름 설정

Server URL

Default is http://localhost:9000

http://[redacted]:8000

가동 중인 소나큐브 주소 작성

Server authentication token

SonarQube authentication token. Mandatory when anonymous access is disabled.

sonarqube-token

위에서 작성한 소나큐브 토큰 Credential 선택

Add

고급

Add SonarQube

## 젠킨스 작업 세팅

연결할 작업 → 구성

Dashboard > dev-back >

상태

변경사항

작업공간

지금 빌드

구성

Project 삭제

Favorite

SonarQube

Rename

Build History

추이

Project dev-back

SonarQube

SonarQube Quality Gate

jenkins-sonar Passed

server-side processing: Success

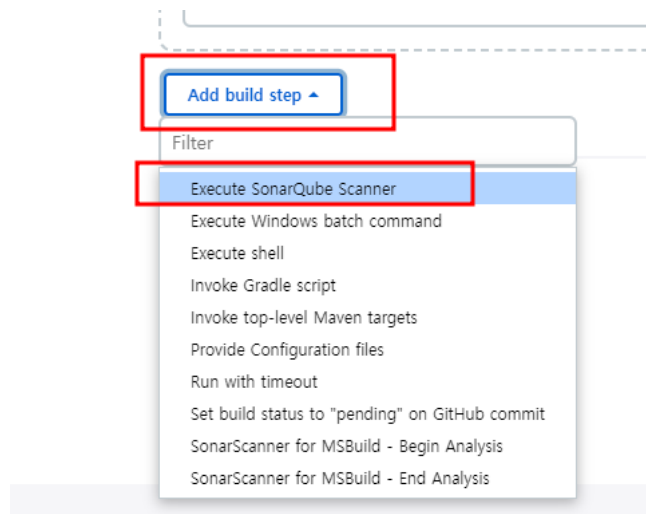
고정링크

Last build, (#213), 2 hr 39 min 전

Last stable build, (#213), 2 hr 39 min 전

Last successful build, (#213), 2 hr 39 min 전

Build steps → Add build step → Execute SonarQube Scanner



Analysis properties 에 작성

```
sonar.login= [발급 받은 소나큐브 키]
sonar.projectKey= [소나 큐브에 등록된 프로젝트 키]
sonar.projectName= [소나 큐브에 등록된 프로젝트 이름]
sonar.host.url= [소나 큐브 접속 url]
sonar.report.export.path=sonar-report.json
detekt.sonar.kotlin.config.path=default-detekt-config.yml
sonar.sources= [젠킨스 컨테이너에 백엔드 프로젝트 경로. src가 위치하는 경로] /src
sonar.java.binaries= [젠킨스 컨테이너에 백엔드 프로젝트 경로. src가 위치하는 경]/src
sonar.java.sourceEncoding=UTF-8
```

위는 스프링 프로젝트 확인 방법

프론트 프로젝트를 확인하려면

젠킨스에 추가로

node.js 를 설치 해주어야 한다.

```
sudo apt-get update
sudo apt-get install nodejs
sudo apt-get install npm
```

sonar큐브로 프론트 코드를 분석하려면 Node.js 14.17 이상 버전이 설치 되어 있어야 한다.

```
sudo apt-get update
sudo apt-get install -y curl
curl -sL https://deb.nodesource.com/setup_14.x | sudo -E bash -
sudo apt-get install -y nodejs
```

위 명령어를 실행하여 최신 버전으로 업데이트

## ▼ 소나 큐브 Jacoco 연동

./build.gradle 파일에 추가 작성

```
plugins {  
    .  
    .  
    .  
    id 'jacoco'  
}  
  
jacoco {  
    // JaCoCo 버전  
    toolVersion = '0.8.10'  
}  
  
.  
.  
.  
  
test {  
    .  
    .  
    .  
    finalizedBy 'jacocoTestReport' // Test 이후 커버리지가 동작하도록 finalizedBy 추가  
}  
  
jacocoTestReport {  
    dependsOn test  
    reports {  
        html.enabled false  
        csv.enabled false  
        xml.enabled true  
  
        xml.destination file("../jacoco/jacoco.xml")  
        //분석리포트 파일명, 파일위치 설정.  
    }  
  
    finalizedBy 'jacocoTestCoverageVerification' // 커버리지 작동 이후 검증하도록 설정  
}
```

## Jenkins 프로젝트 구성 세팅

Command

See [the list of available environment variables](#)

```

./start-dev.sh

#docker-compose -f docker-compose-sonar.yml stop
#docker-compose -f docker-compose-sonar.yml build

docker cp dev-back_backend_1:./app.jar /var/jenkins_home/workspace/dev-back/app/build/
docker-compose -f docker-compose-sonar.yml up -d
#docker start dev-back_backend_1

cd back-end/roughcode
chmod +x ./gradlew
# ./gradlew clean test
./gradlew jacocoTestReport
# ./gradlew build -x test

```

```

# gradlew 가 있는 위치로 이동
cd back-end/roughcode
chmod +x ./gradlew
./gradlew jacocoTestReport

```

## Sonarqube - Analysis properties에 추가

```

sonar.java.coveragePlugin = jacoco
sonar.coverage.jacoco.xmlReportPaths= /var/jenkins_home/workspace/dev-back/back-end/jacoco/jacoco.xml
# 젠킨스 컨테이너 내의 프로젝트에 jacoco.xml 이 생성되는 위치로 설정

```