모이다 포팅 메뉴얼

> SSAFY 8기 특화 프로젝트, C207 지꿈지꿈

1. 프로젝트 개요

✅ 서비스 명 : 모이다

✅ 한 줄 소개 : 야생동물 상생 기부 플랫폼

2. 프로젝트 사용 도구

이슈 관리 : JIRA

형상 관리 : Gitlab

소통 : Mattermost, Discord, Notion, Kakaotalk

배포 : Docker, Jenkins

3. 개발 환경

Frontend

```
VSCode IDE : v1.77.0
React : v18.2.0
React-Query : v4. 28. 0
React Web3 : v1.9.0
Axios : v1. 3. 4
Styled-Components : v5. 3. 8
Tailwild : v1. 1. 7
Node.js : v18. 15. 0
```

Backend

```
IntelliJ IDEA : v2022. 03
SpringBoot : v3. 0. 4
JDK : v17. 0. 6
Spring Security : v3. 0. 4
JWT : v0. 11. 5
JPA : v3. 0. 4
Swagger : v3. 0. 0
MariaDB : v10. 11. 2
Redis : v7. 0. 10
S3 : v2. 2. 6
```

Smart Contract(Blockchain, NFT)

```
Truffle : v5. 8. 1
Ganache : v7. 7. 7
Truffle Web3 : v1. 8. 2
Solidity : v0. 8. 19
Metamask : v10. 27. 0
```

CI/CD

```
AWS EC2
Nginx
Docker
Jenkins
Gitlab
```

4. 외부 서비스

1. AWS EC2, S3 : S3Config에 해당 설정 내용 있음
2. Redis : RedisConfig에 해당 설정 내용 있음
3. Swagger : SwaggerConfig에 해당 설정 내용 있음

5. 빌드 시 사용되는 환경 변수

```
ENV REACT_APP_API_URL http://back
ENV REACT_APP_SEPOLIA_API_URL=https://sepolia.infura.io/v3/a4636fca95804d9ab953276a4ea93748
ENV REACT_APP_SEPOLIA_ADMIN_PUBLIC_KEY=0xCf779DB49Fc0CA0Ef3ba47E12Cf0b25b195879c5
ENV REACT_APP_SEPOLIA_ADMIN_PRIVATE_KEY=b0148045d850bc0d896cf9b99603fcfdde940b61daf07d5c93c20
ENV REACT_APP_SEPOLIA_TOKEN_CONTRACT=0x735A38151F906EB5E7873491E57c447b3ca9Ac49
ENV REACT_APP_PINATA_JWT=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySW5mb3JtYXRpb24iOnsiaWQ
```

```
REACT_APP_API_URL : 프론트엔드 proxy path
REACT_APP_SEPOLIA_API_URL : Infura에서 제공받은 세폴리아 테스트넷 url
REACT_APP_SEPOLIA_ADMIN_PUBLIC_KEY : 컨트랙트 배포 계정
REACT_APP_SEPOLIA_ADMIN_PRIVATE_KEY : 컨트랙트 배포 계정의 개인키
REACT_APP_SEPOLIA_TOKEN_CONTRACT : 컨트랙트 계약 토큰
REACT_APP_PINATA_JWT : 피나타 인증 토큰
```

6. application.yml

MariaDB

```
spring:
  datasource:
    password: root
    username: root
    url: jdbc:mariadb://mariadb:3306/moida?useSSL=false
    driver-class-name: org.mariadb.jdbc.Driver
```

Redis

```
data:
    redis:
      host: redis
      port: 6379
```

mail

```
mail:
    host: smtp.gmail.com
    port: 587
    username: jikumjikum207@gmail.com
    password: kmueafliiwvcdmfx
    properties:
      mail.smtp.auth: true
      mail.smtp.starttls.enable: true
```

Swagger

```
springdoc:
  api-docs:
    enabled: true
  swagger-ui:
    path: /swagger-ui.html
    disable-swagger-default-url: true
    display-request-duration: true
    tags-sorter: alpha
    operations-sorter: alpha
    doc-expansion: none
    syntax-highlight:
      theme: nord
    urls-primary-name: TEST API
    persist-authorization: true
    query-config-enabled: true
  pre-loading-enabled: true
  packages-to-scan: com.ssafy.moida
```

AWS S3

```
cloud:
  aws:
    s3:
      bucket: moida.bucket
    region:
      static: ap-northeast-2
      auto: false
    stack:
      auto: false
    credentials:
      access-key: PUBLIC 키
      secret-key: PRIVATE 키
```

docker-compose.yml

```
version: "3.3"

services:
  redis:
    image: redis
    container_name: redis
    ports:
      - 6379:6379
    networks:
      - moida
    restart: always
  front:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    ports:
      - 80:80
    depends_on:
      - back
    container_name: front
    networks:
      - moida
  back:
```

```
    build:
      context: ./backend
      dockerfile: Dockerfile
    ports:
      - 5000:5000
    container_name: back
    depends_on:
      - redis
    networks:
      - moida

networks:
  moida:
    external:
      name: moida
```

7. 로컬 빌드

1) 프론트엔드 : 터미널 명령어 실행

```
npm install
npm start
```

2) 백엔드 : MoidaApplication 실행

8. 배포

1) 프론트엔드 배포

```
# node 이미지를 받는다 build이미지이구나
# node 기반의 이미지로 생성된다.
FROM node:latest as build-stage

# RUN, CMD, ENTRYPOINT의 명령이 실행될 디렉터리
WORKDIR /app

# 로컬 proxy path 설정
ENV REACT_APP_API_URL http://back
ENV REACT_APP_SEPOLIA_API_URL=https://sepolia.infura.io/v3/a4636fca95804d9ab953276a4ea93748
ENV REACT_APP_SEPOLIA_ADMIN_PUBLIC_KEY=0xCf779DB49Fc0CA0Ef3ba47E12Cf0b25b195879c5
ENV REACT_APP_SEPOLIA_ADMIN_PRIVATE_KEY=b0148045d850bc0d896cf9b99603fcfdde940b61daf07d5c93c20cf675ac575f
ENV REACT_APP_SEPOLIA_TOKEN_CONTRACT=0x735A38151F906EB5E7873491E57c447b3ca9Ac49
ENV REACT_APP_PINATA_JWT=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySW5mb3JtYXRpb24iOnsiaWQiOiI0ZThlMjBjNC1jNmJjLTQwMGEtOGEwMC1mYzU1

# package*.json이 WORKDIR에 복사된다
COPY package*.json ./

# 복사했으니 디펜던시 설치가 가능하다
RUN npm install

# 소스코드를 복사한다
COPY ./ .

# ENV NODE_OPTIONS=--max_old_space_size=2048

# build 해서 /dist 폴더에 빌드 파일이 생성된다.
RUN npm run build

# 3000번 포트 노출
EXPOSE 80

# npm start 스크립트 실행
# CMD ["npm", "start"]
# CMD ["nginx", "-g", "daemon off;"]

# nginx 이미지를 받는다. 실행 이미지이구나
FROM nginx:1.23.4-alpine

COPY ./conf /etc/nginx/
```

```
# builder에서 빌드한 바이너리를 실행할 이미지로 전달해주기 위해 copy --from 옵션을 사용하여 실행 이미지로 전달한다
COPY --from=build-stage /app/build /usr/share/nginx/html
```

2) Nginx 배포(frontend)

nginx.conf

```
user  nginx;
worker_processes  auto;

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

events {
    worker_connections  1024;
}

http {
    client_max_body_size 50M;
    include       /etc/nginx/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  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*.conf;
    server_names_hash_bucket_size 64;
}
```

conf/default.conf

```
upstream back {
    server back:5000;
}

server {
    listen 80;
    server_name  http://j8c2071.p.ssafy.io;

    location / {
        root  /usr/share/nginx/html;
        index index.html index.htm;
        try_files $uri $uri/ /index.html;
    }

    location /api {
        proxy_pass http://back/api;
    }
}
```

3) 데이터베이스 배포

```
docker pull mariadb:latest
docker run -p 3307:3306 --name mariadb -e MARIADB_ROOT_PASSWORD=root -d mariadb
docker exec -it mariadb bash
mysql -u root -p
```

```
create database moida;
use moida;
```

4) 백엔드 배포

```
FROM gradle:8.0.2-jdk17 as builder
WORKDIR /build

# 그래들 파일이 변경되었을 때만 새롭게 의존패키지 다운로드 받게함.
COPY build.gradle settings.gradle /build/
RUN gradle build -x test --parallel --continue  > /dev/null 2>&1 || true

# 빌더 이미지에서 애플리케이션 빌드
COPY . /build
RUN gradle clean build -x test --parallel

# APP
FROM openjdk:17.0-slim
WORKDIR /build

# 빌더 이미지에서 jar 파일만 복사
COPY --from=builder /build/build/libs/moida-0.0.1-SNAPSHOT.jar .

EXPOSE 5000

# root 대신 nobody 권한으로 실행
ENTRYPOINT [                                                \
    "java",                                                 \
    "-jar",                                                 \
    "-Djava.security.egd=file:/dev/./urandom",              \
    "-Dsun.net.inetaddr.ttl=0",                             \
    "moida-0.0.1-SNAPSHOT.jar"              \
]
```

9. 젠킨스 배포

1) ubuntu 세팅

```
// 설치 되어있는 패키지의 새로운 버전이 있는지 확인
sudo apt-get update

// 패키지 버전 업그레이드
sudo apt-get -y upgrade

// node.js 컴파일 위해 build-essential 설치
sudo apt-get install build-essential

// 서버에 API 요청 위한 curl 설치
sudo apt-get install curl

/*
 * -s -> 에러 같은 정보 보여주지 않는 옵션
 * -L -> 서버에게서 3xx 같은 리다이렉션 응답이 오면 해당 페이지로 다시 요청 보내는 옵션
 * node.js LTS 최신 버전 설치 위해 LTS 16 버전으로 setup
*/
curl -sL https://deb.nodesource.com/setup_16.x | sudo -E bash --
```

2) Docker 세팅

```
sudo apt-get update

sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
```

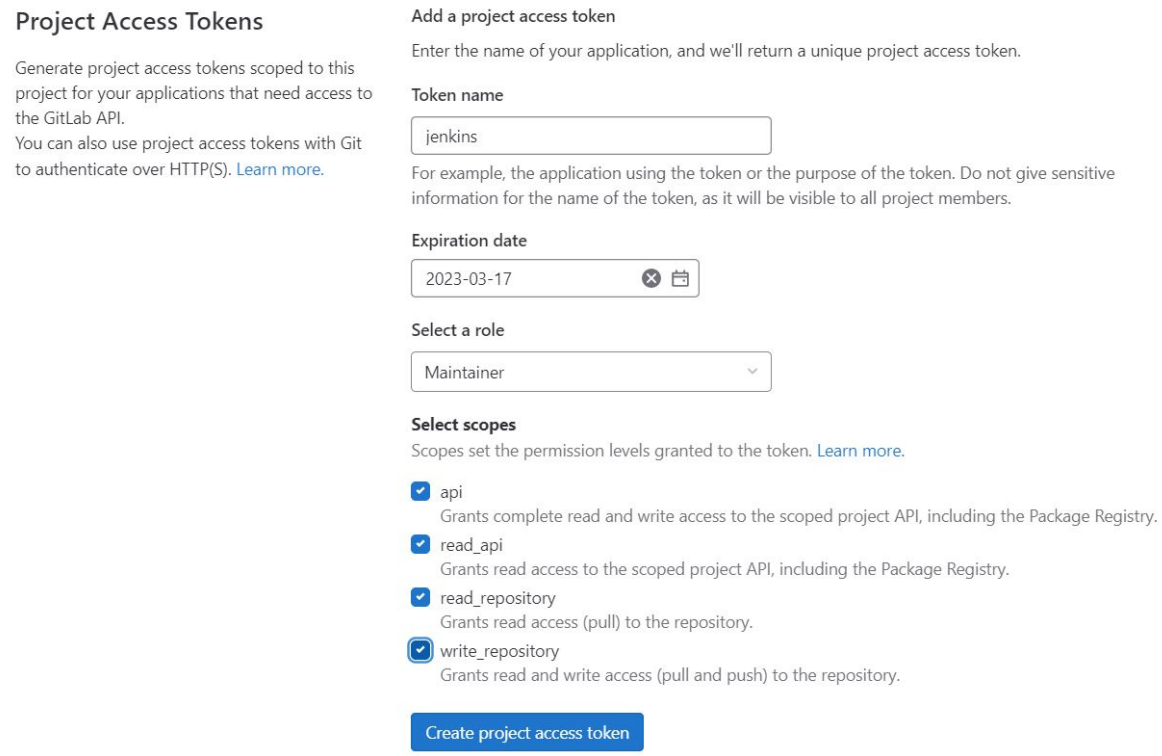
```
    software-properties-common

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo apt-key fingerprint 0EBFCD88

sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"

sudo apt-cache policy docker-ce
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io

// 명령어로 도커가 설치되었는지 확인한다
sudo docker

/*
sudo 없이 도커 명령어를 사용하기 위해
자신의 ec2 사용자 이름을 docker 그룹에 추가해야 한다.
exit 후 ec2 에 다시 재접속 하면 sudo 없이 도커 명령어를 사용할 수 있다.
*/
sudo usermod -aG docker (ec2 사용자이름)
```

3) jenkins 컨테이너 등록 및 실행

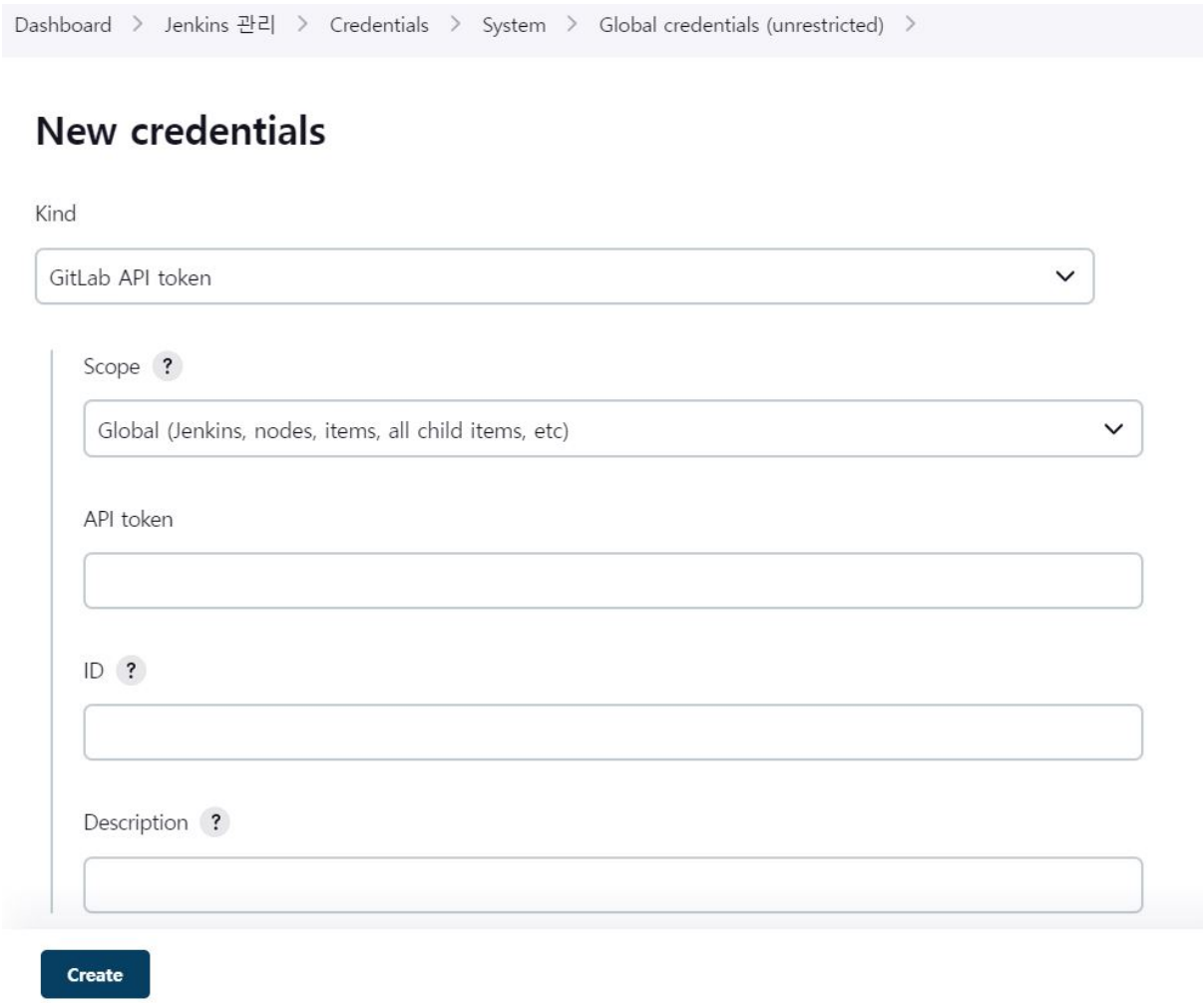
```
// jenkins/jenkins로 젠킨스 최신 이미지를 설치한다.
docker run -d -p 8080:8080 -v /home/jenkins:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock --name jenkins -u root jenk

// 컨테이너 실행
sudo docker start jenkins

// 접속을 위한 로그 확인
sudo docker logs jenkins
```

4) jenkins 연결

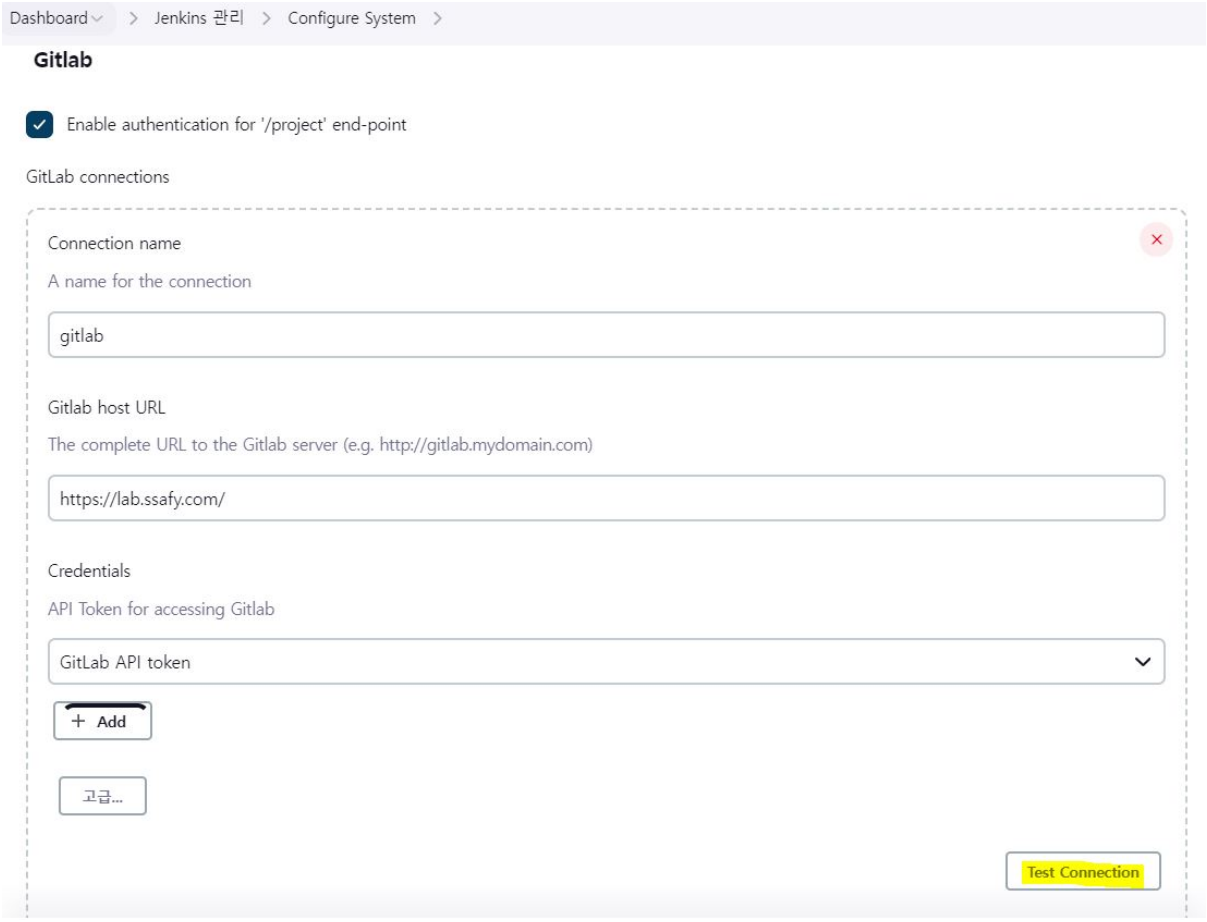
1️⃣ gitlab에서 Settings → token 발급



2️⃣ jenkins 관리 → Manage Credentials 이동 → global에서 Add credentials 클릭 → GitLab API token 등록

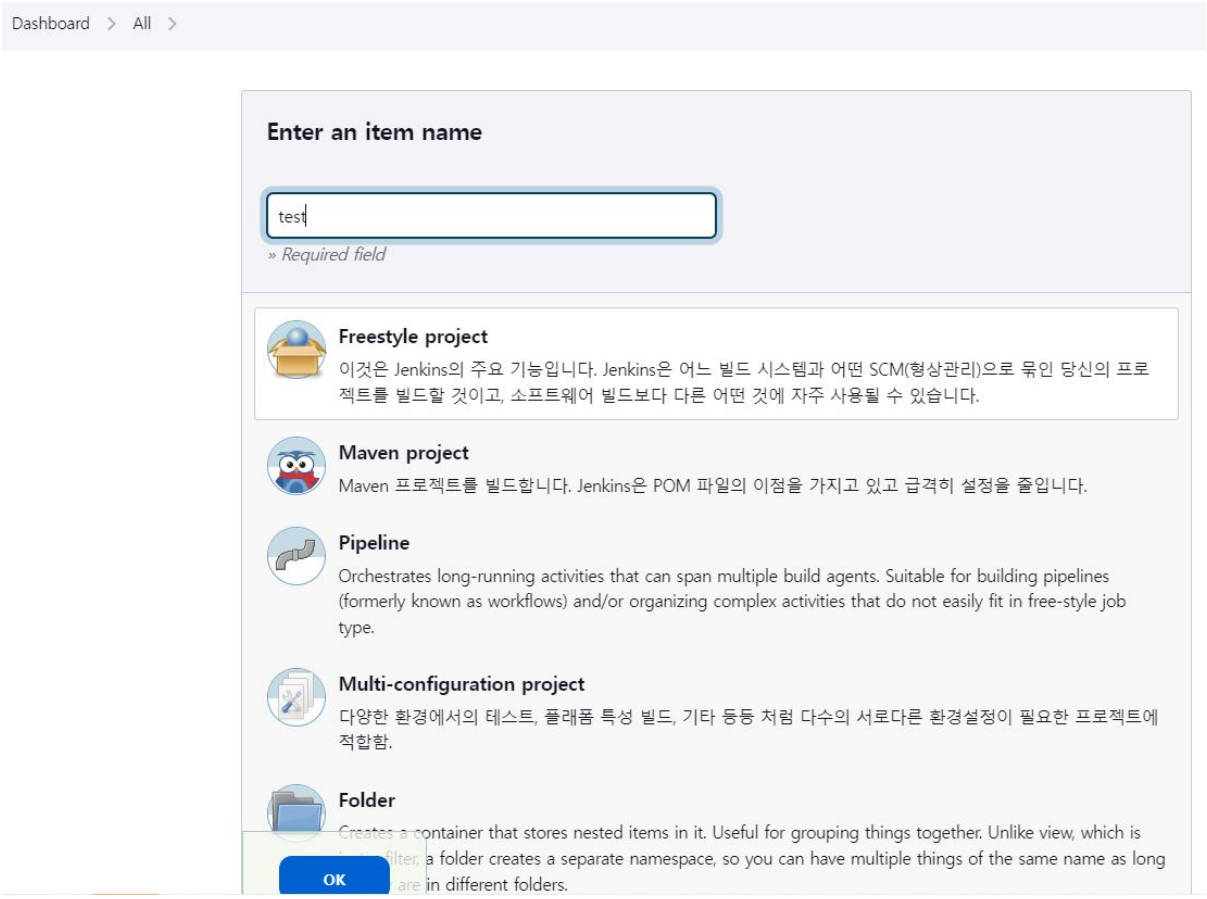


3️⃣ Jenkins 관리 → 시스템 설정 → Gitlab



✅ Repository 연결

1️⃣ item 생성 : gitlab(이름 아무거나) → Freestyle project 선택



2️⃣ Credential 추가 : 저장소 연동하기 위한 용도(+ 아래에 연결할 gitlab의 신원확인 용)

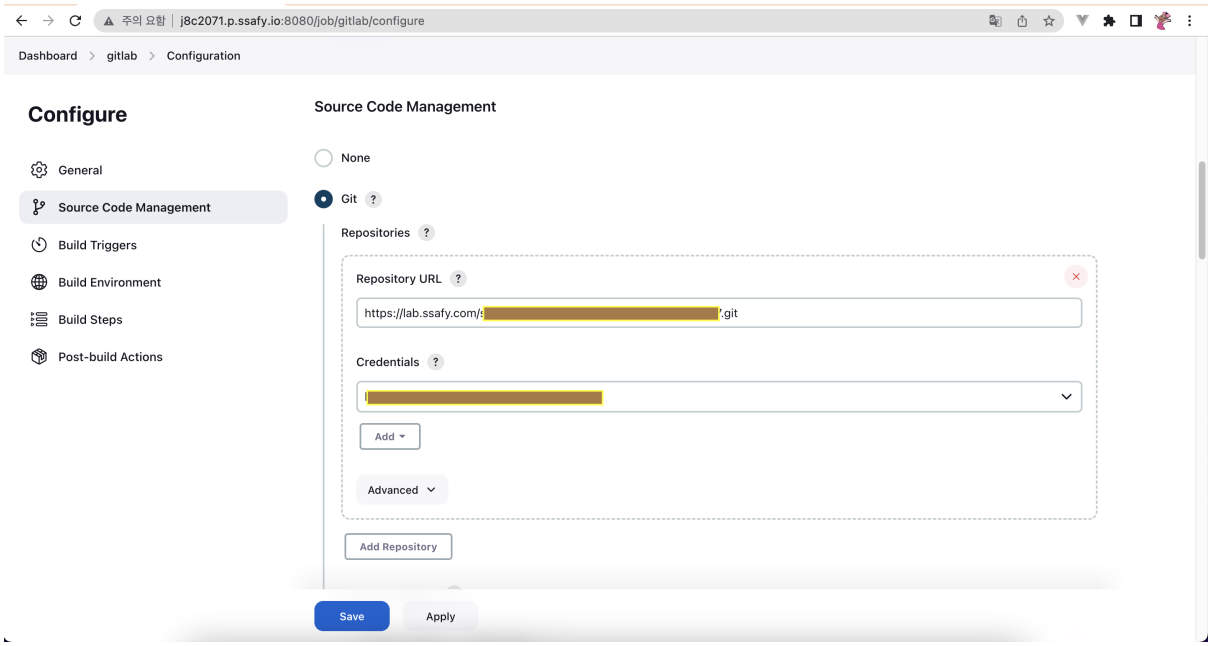
⇒ Credentials → Add Jenkins

```
Kind: Username with password
Scope: Global
Username: [깃랩 로그인할 때 쓰는 이메일]
Password: [gitlab에서 받은 access token 값]
```

✅ item(아까 생성한 gitlab) 환경설정

1️⃣ gitlab → Configure → Source Code Management 이동

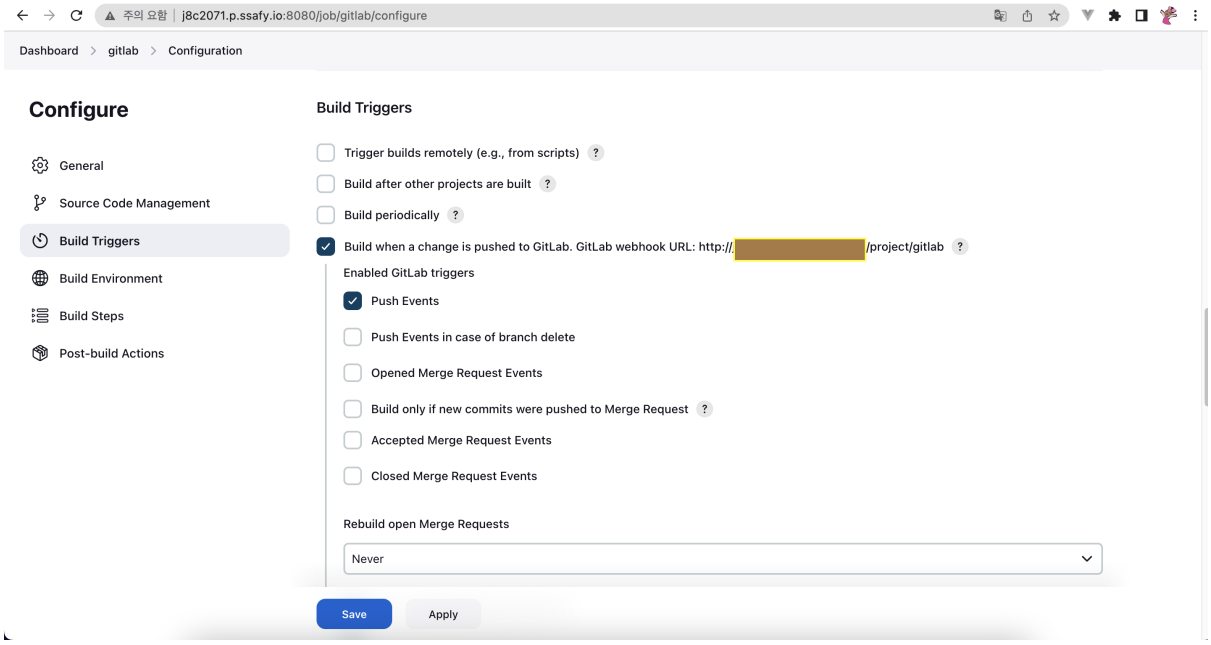
⇒ Git 선택 후 Repository URL : 깃랩 프로젝트 Clone url & Credentials : 바로 위에서 추가한 Credential 선택



⇒ Branches to build → Branch Specifier에 */develop 기입 (자동 배포 시킬 브랜치 이름)

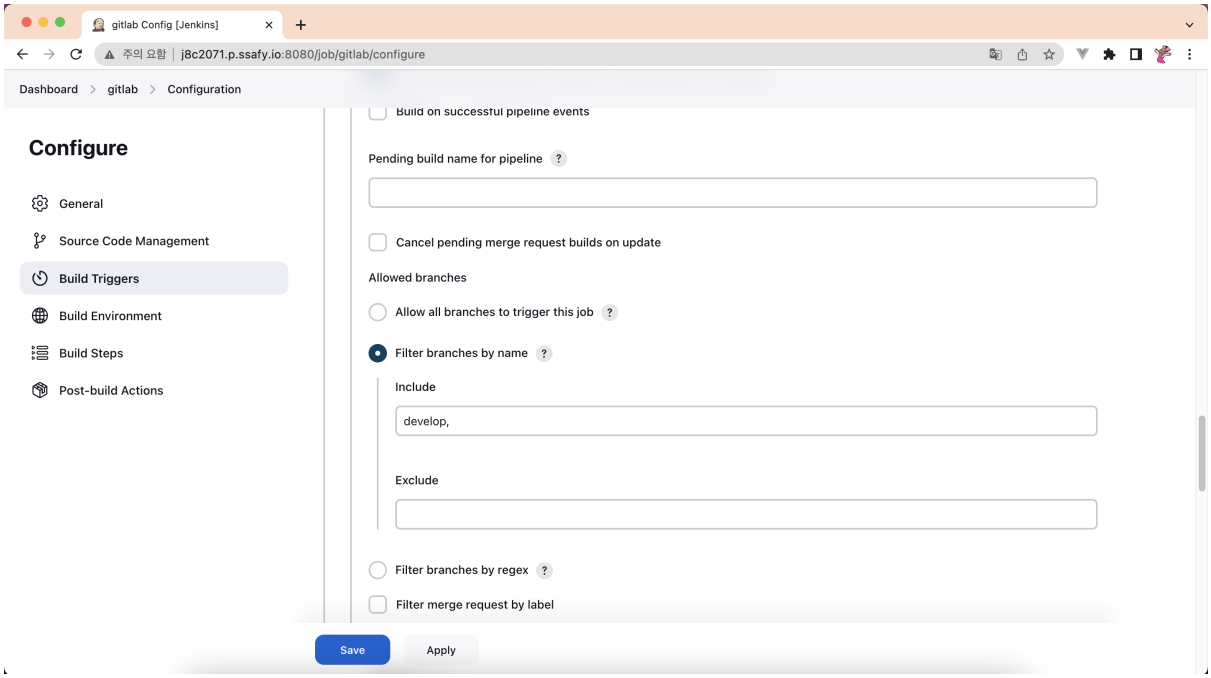
2️⃣ Build trigger(빌드 유발) 설정

→ Build when a change is pushed to Gitlab 체크 → GitLab webhook URL 복사 → Push Events 체크

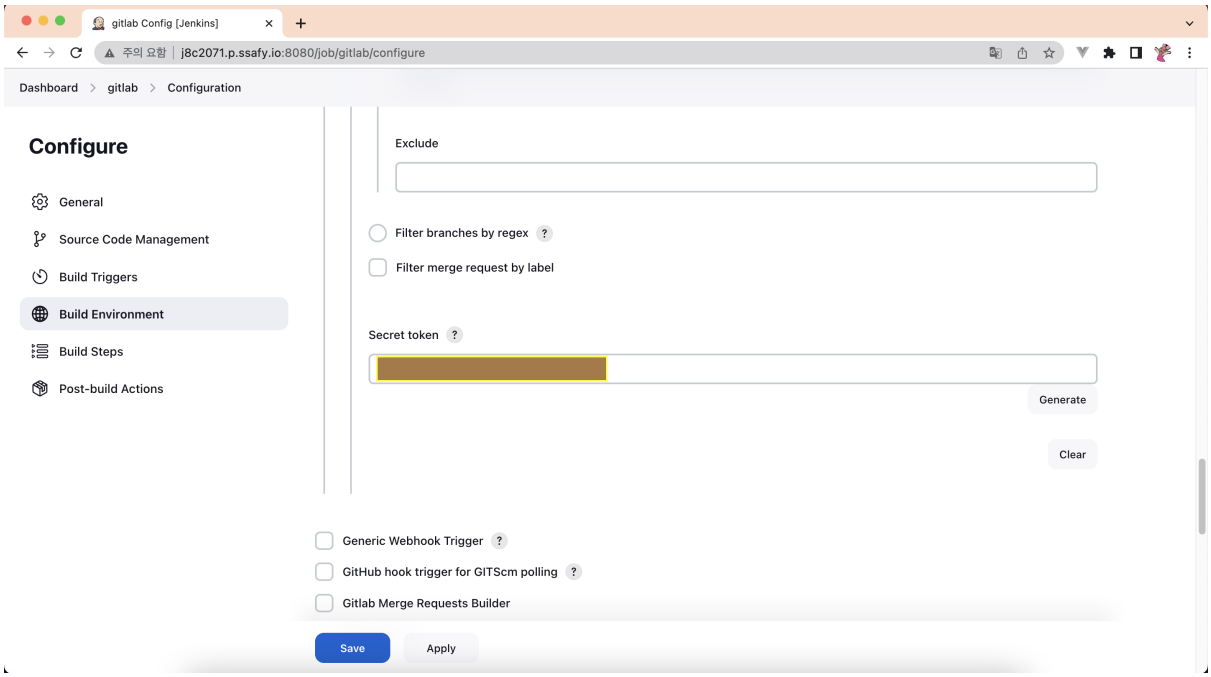


Merge Open은 build 안하고 develop 브랜치에 push 할 때에만 빌드하게 설정했음다

3️⃣ 빌드 유발 → 고급(Advance) → Allowed branches 에 Branch filter 를 걸어서 develop만 철저히 감시하게 설정



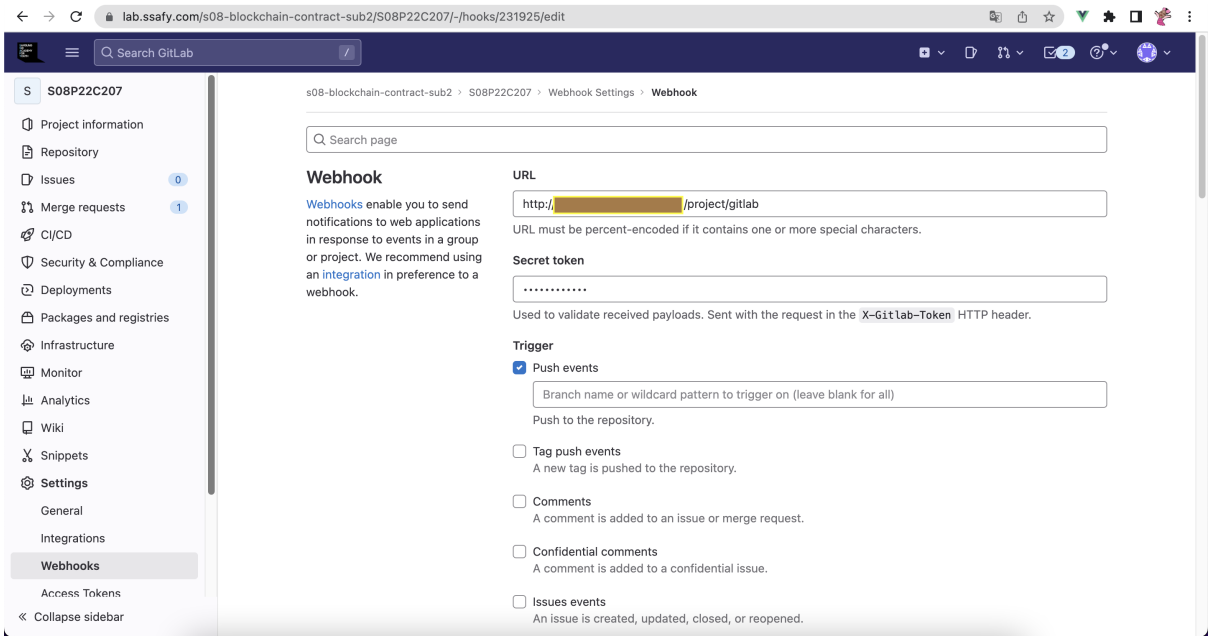
4️⃣ 빌드 유발 → 고급(Advance) → {Secret token → Generate 후 복사} → 저장(Save)

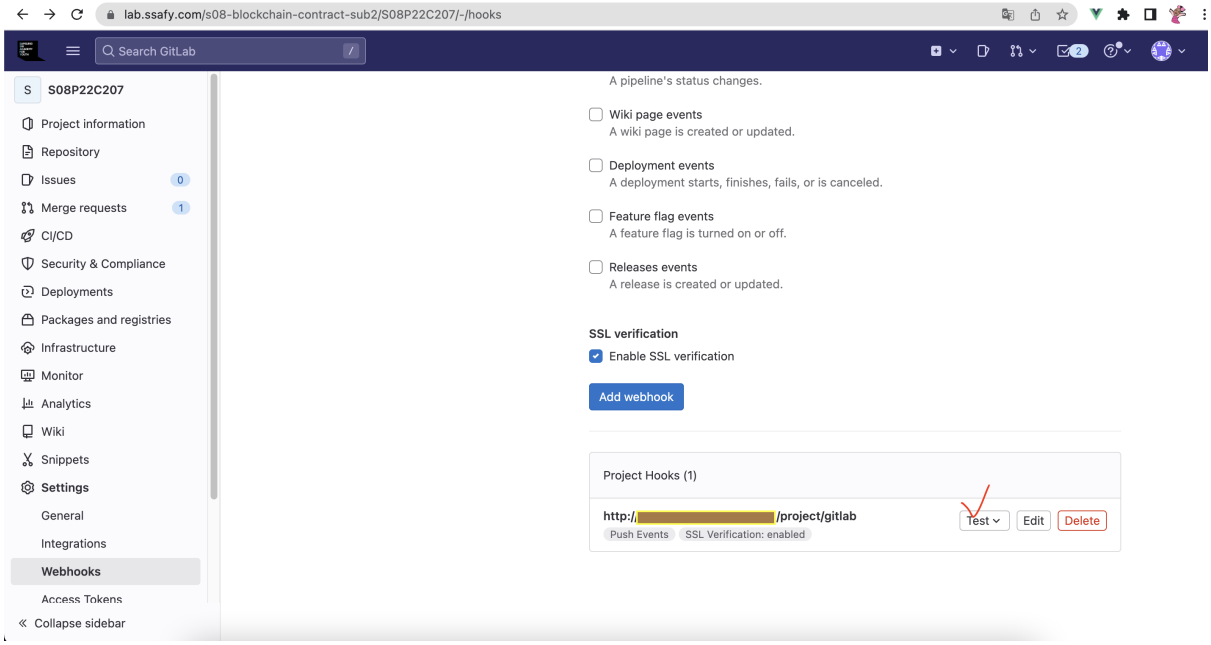


✅ WebHook 연결

프로젝트 gitlab 으로 이동 → Settings → WebHook

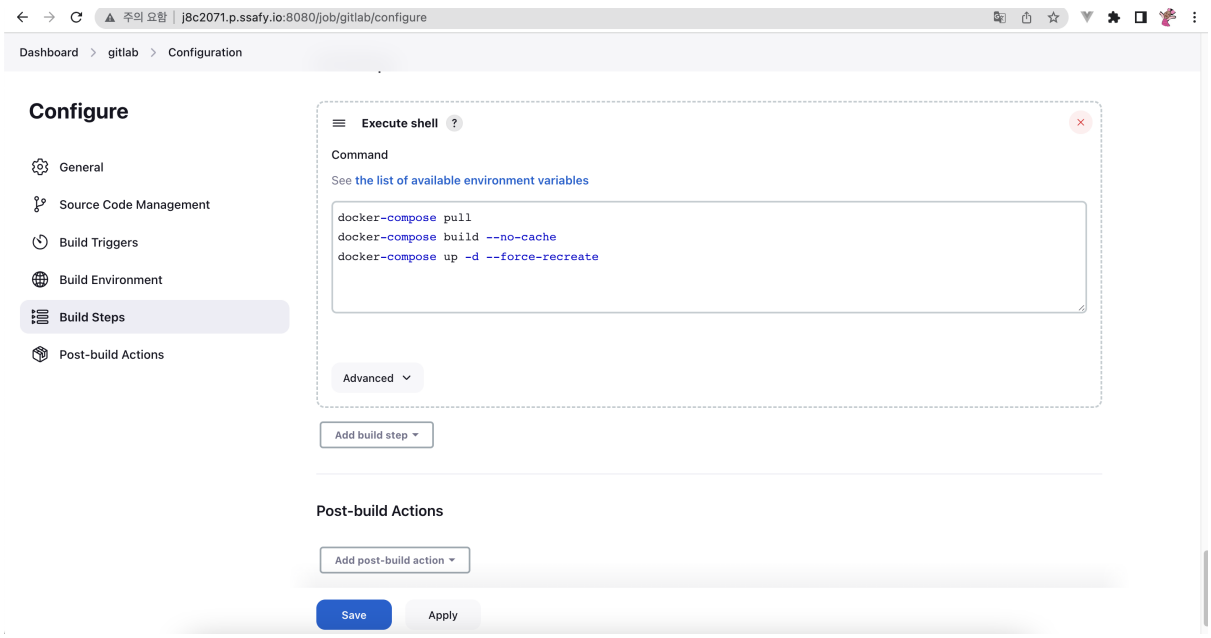
⇒ secret token 에 아까 복사한 값 붙여넣기 → Push Event만 체크





생성 후 Test 눌러서 제대로 연결되었는지 확인 가능

✅ 자동 배포 스크립트 작성

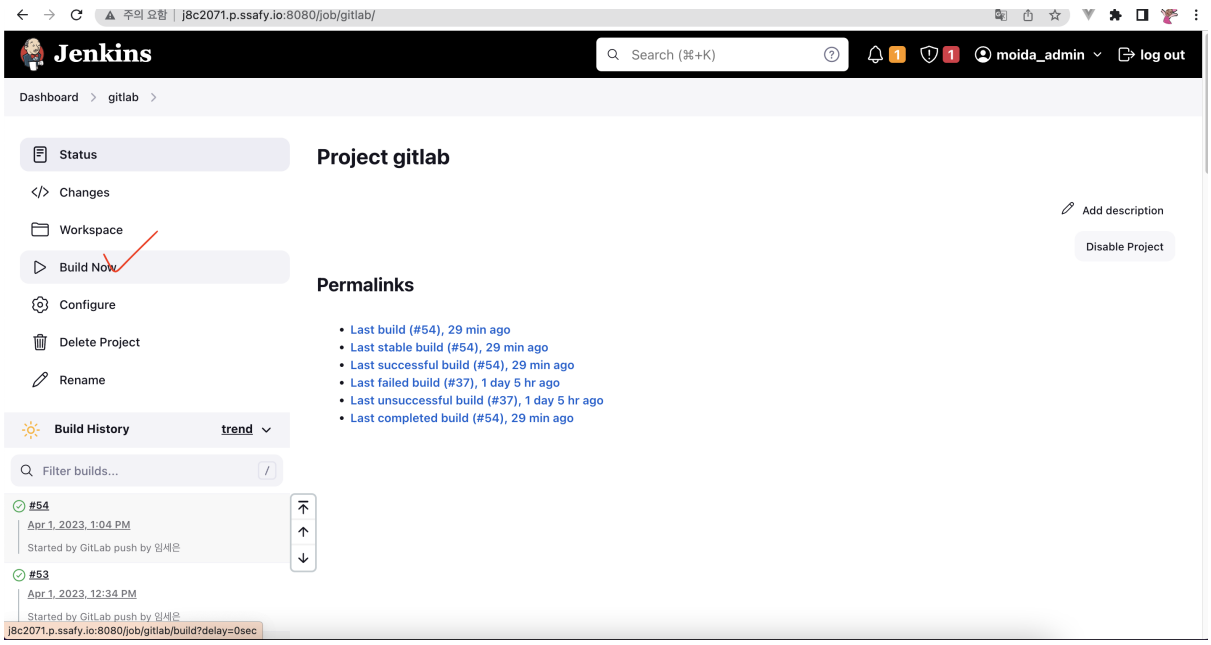


이렇게 작성해도 되고

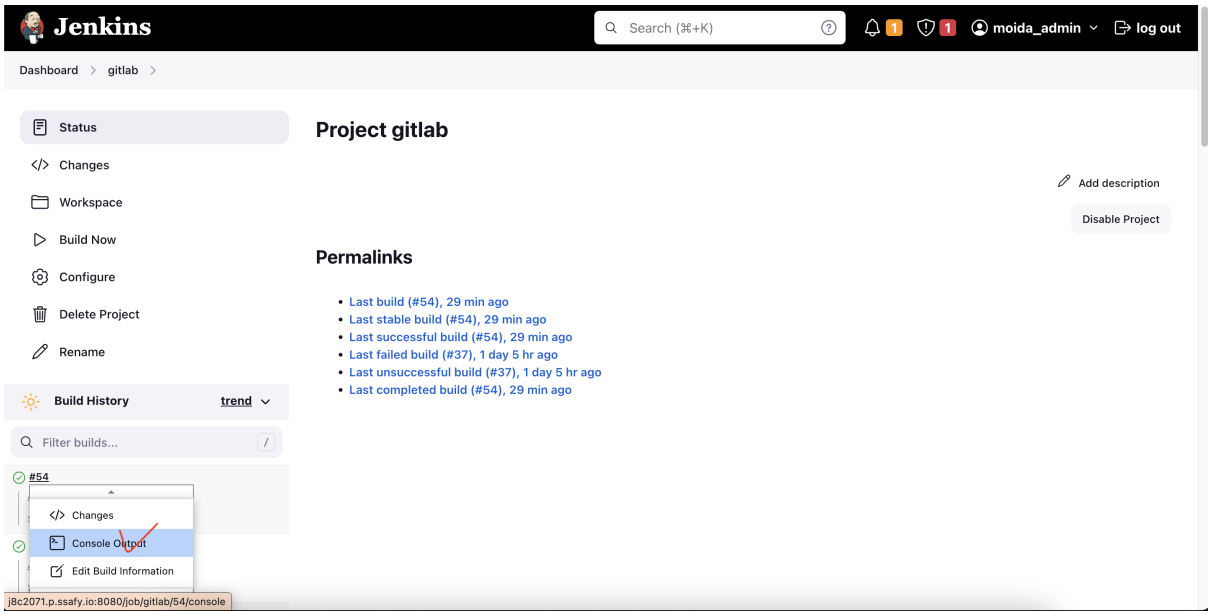
```
// 이렇게 작성해도 됨
docker-compose down
docker-compose pull
docker-compose build --no-cache
docker-compose up -d back
docker-compose up -d front
```

✅ 젠킨스 build 테스트

gitlab 에 들어가서 Build Now 누르기



Build History에서 Console Output을 확인할 수 있음!



10. 설정 파일

S3Config.java

```
package com.ssafy.moida.config;

import com.amazonaws.auth.AWSStaticCredentialsProvider;
import com.amazonaws.auth.BasicAWSCredentials;
import com.amazonaws.services.s3.AmazonS3Client;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * [세은] S3 설정 Config
 */
@Configuration
public class S3Config {
    @Value("${cloud.aws.credentials.access-key}")
    private String iamAccessKey;
    @Value("${cloud.aws.credentials.secret-key}")
    private String iamSecretKey;
    @Value("${cloud.aws.region.static}")
    private String region;
    @Bean
    public AmazonS3Client amazonS3Client(){
        BasicAWSCredentials awsCredentials = new BasicAWSCredentials(iamAccessKey, iamSecretKey);
        return (AmazonS3Client) AmazonS3ClientBuilder.standard()
            .withRegion(region).enablePathStyleAccess()
            .withCredentials(new AWSStaticCredentialsProvider(awsCredentials))
            .build();
    }
}
```

RedisConfig.java

```
package com.ssafy.moida.config;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
```

```
import org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.data.redis.serializer.StringRedisSerializer;

@Configuration
public class RedisConfig {
    @Value("${spring.data.redis.host}")
    private String host;

    @Value("${spring.data.redis.port}")
    private int port;
    /*
     * Redis 서버와의 통신을 위한 low-level 추상화를 제공
     * 설정에 따라서 새로운 RedisConnection 또는 이미 존재하는 RedisConnection을 리턴
     * */
    @Bean
    public RedisConnectionFactory redisConnectionFactory() {
        return new LettuceConnectionFactory(host,port);
    }

    /*
     * Redis 서버에 Redis Command를 수행하기 위한 high-level 추상화를 제공
     * Redis 서버에 데이터 CRUD를 위한 Key Type Operations 와 Key Bound Operations 인터페이스를 제공
     * */
    @Bean
    public RedisTemplate<String, Object> redisTemplate() {
        RedisTemplate<String, Object> redisTemplate = new RedisTemplate<>();
        redisTemplate.setConnectionFactory(redisConnectionFactory());
        redisTemplate.setKeySerializer(new StringRedisSerializer());
        redisTemplate.setValueSerializer(new StringRedisSerializer());
        return redisTemplate;
    }

    @Bean
    public StringRedisTemplate stringRedisTemplate() {
        StringRedisTemplate stringRedisTemplate = new StringRedisTemplate();
        stringRedisTemplate.setKeySerializer(new StringRedisSerializer());
        stringRedisTemplate.setValueSerializer(new StringRedisSerializer());
        stringRedisTemplate.setConnectionFactory(redisConnectionFactory());
        return stringRedisTemplate;
    }

}
```

SwaggerConfig.java

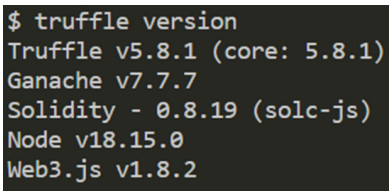
```
package com.ssafy.moida.config;

import io.swagger.v3.oas.models.Components;
import io.swagger.v3.oas.models.OpenAPI;
import io.swagger.v3.oas.models.info.Info;
import io.swagger.v3.oas.models.security.SecurityScheme;
import io.swagger.v3.oas.models.security.SecurityScheme.In;
import io.swagger.v3.oas.models.security.SecurityScheme.Type;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 * [세은] Swagger Api Config
 */
@Configuration
public class SwaggerConfig {
    @Bean
    public OpenAPI openAPI(){
        Info info = new Info()
            .title("모이다 API Documentation")
            .version("v1.0.0")
            .description("모이다 API에 대한 설명 문서입니다!");

        SecurityScheme securityScheme = new SecurityScheme()
            .type(Type.HTTP)
            .scheme("Bearer")
            .bearerFormat("JWT")
            .in(In.HEADER).name("Authorization");

        return new OpenAPI()
            .components(new Components().addSecuritySchemes("bearerAuth", securityScheme))
            .info(info);
    }
}
```

11. 스마트 컨트렉트 배포

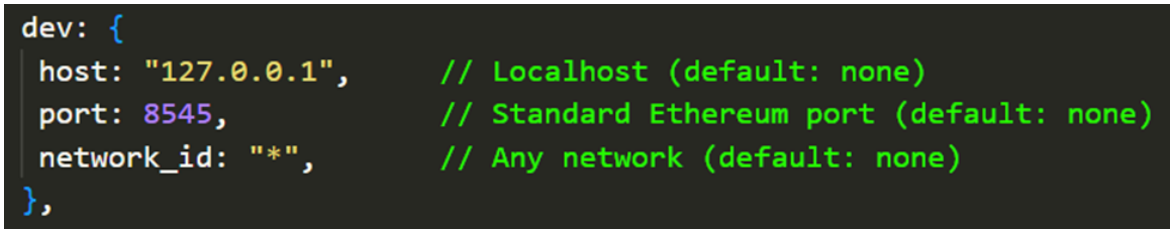
해당 메뉴얼은 Truffle 기준으로 작성됨

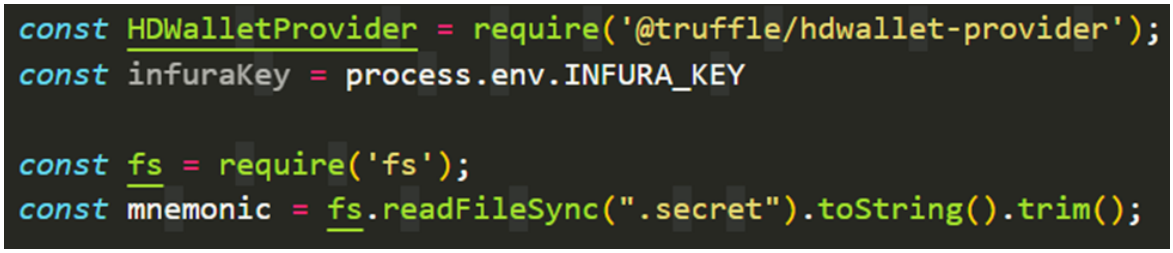
1. **Truffle 설치 및 확인**

```
npm install -g truffle
truffle version
```



2. **Truffle-config 설정 : 스마트 컨트렉트를 배포할 네트워크 설정**






```
secret : ADMIN의 니모닉 구문
SEPOLIA_API_URL : Infura에서 제공받은 세폴리아 테스트넷 url
SEPOLIA_ADMIN_PUBLIC_KEY : 컨트랙트 배포 계정
SEPOLIA_ADMIN_PRIVATE_KEY : 컨트랙트 배포 계정의 개인키
```

3. **컨트렉트 배포**

   만약, 로컬 호스트에 배포를 하려변 ganache-cli를 설치한 후, 로컬 서버를 먼저 열어야 함

```
npm install -g ganache-cli
ganache-cli
배포 커맨드 : truffle migrate –network { 네트워크명 }
```