포팅 메뉴얼: WALK_WALK

> SSAFY 10기 특화 프로젝트 구미 2반 10팀

0. 목차

1. 프로젝트 기술 스택

- Frontend
  - Visual Studio Code (IDE) `1.86.2`
  - HTML5, CSS3, JavaScript(ES6)

    - React `18.2.0`
    - Stompjs `7.0.0`
    - zustand `4.5.2`
    - Nodejs `20`
    - Vite `5.0.0`
    - Tailwind CSS `3.4.1`
    - Web3js `4.0.1`
    - Solidity `0.8.25`
    - 사용한 외부 API
        - Web-Speech-API
        - Google-Fitness-API
        - Google-geoLocation-APl (GPS 추적용)
        - Google-Map-React-APl

- Backend
    - Intellij `2023.3.2`
    - JVM Open JDK `17`
    - JWT `0.12.4`
    - Spring Boot `3.0.13`
        - JAVA Spring Data JPA
        - Spring Security
        - SSEEmitter
    - OAuth2.0 `6.8.0`
    - STOMP-WEB-SOCKET `2.3.4`
    - REDIS Lettuce `6.3.2`
    - REDIS `7.2.4`
    - Gradle
    - AWS S3 Bucket Cloud `2.2.6`
    - AWS Transcribe `1.12.691`
    - AWS Lambda
    - Spring Batch `5.1.1`
    - Web3j `4.11.2`
    - Google TTS CLOUD AI

- CI/CD
    - AWS EC2
        - Nginx `1.18.0`

- Ubuntu `20.04 LTS`
- Docker `25.0.2`
- Jenkins `2.443`

- Docker Hub

2. 환경 변수 설정

(1) FrontEnd

> .env 라는 이름으로 파일명을 작성하고, 아래 내용을 기입합니다.

```
VITE_GOOGLE_CLIENT_ID= "본인의 구글 OAuth2 클라이언트 아이디"
VITE_GOOGLE_CLIENT_SECRET="본인의 구글 OAUth2 클라이언트 Secret"
VITE_REDIRECT_URI_DEV="배포 서버 Redirect URL"
VITE_REDIRECT_URI_PROD"로컬 서버 Redirect URL"
VITE_NODE_ENV=local // 환경 토글용 Value 로컬 실행 시 Local, 배포 실행 시 DEV
```

(2) BackEnd

> Backend 설정 파일은 application.properties 를 삭제하고, yaml 파일로 통일한다. 사용한 application.yml 파일은 다음과 같습니다.

가. application.yml

```yaml
spring:
  profiles:
    active: dev # 어떤 설정파일을 쓸지에 대한 표시
  servlet:
    multipart:
      enabled: true # 멀티파트 업로드 지원 여부 (default: true)
      max-file-size: 100MB
      max-request-size: 100MB

  output: # 콘솔 로그에 색깔 입히기 ♡ㅅ(ˆᗜˆ ू )ノ♡
    ansi:
      enabled: always

  cache:
    type: redis
  data:
    # redis 설정
    redis:
      lettuce:
        pool:
          max-active: 5 # pool에 할당될 수 있는 커넥션 최대수
          max-idle: 5 # pool의 'idle' 커넥션 최대수
          min-idle: 2
      host: ${REDIS_HOST}
      port: 6379
      password: ${REDIS_PASSWORD}
  jackson:
    serialization:
```

```
      fail-on-empty-beans: false

  batch:
    job:
      enabled: false # 키자마자 자동 실행 방지
    jdbc:
      initialize-schema: always




google:
  client-id: ${GOOGLE_CLIENT_ID}
  client-secret: ${GOOGLE_SECRET}
  local-redirect-uri: http://localhost:8081/oauth2/callback/google
  dev-redirect-uri: https://j10d210.p.ssafy.io/oauth2/callback/google
  token-uri: https://oauth2.googleapis.com/token
  user-info-uri: https://www.googleapis.com/oauth2/v2/userinfo
  authorization-uri: https://accounts.google.com/o/oauth2/auth?
client_id=${GOOGLE_CLIENT_ID}&redirect_uri=${GOOGLE_REDIRECT_URI}&response_type=c
ode&scope=https://www.googleapis.com/auth/userinfo.email
https://www.googleapis.com/auth/userinfo.profile
  Text-to-Speech-key: ${Google_TTS_CLOUD_API_KEY}
server:
  port: 8081


cloud:
  aws:
    credentials:
      access-key: ${S3_ACCESS_KEY}
      secret-key: ${S3_SECRET_KEY}
    region:
      static: ap-northeast-2  # S3가 가용될 수 있는 지역 -> 동북아시아/대한민국으로 한정
    stack:
      auto: false # 프로젝트를 EC2로 배포 시에, 이 값이 true 되어있으면 SpringBoot에서
CloudFormation 구성을 시작함
      # 근데 따로 설정한 CloudFormation이 없으면 프로젝트가 배포단계에서 죽어버림
      # 따라서 우리는 애초에 Cloud 구성 기능을 사용하지 않겠다는 의미에서 false로 바꿔 둔
다.

    s3:
      bucket: d210

jwt:
  header: Authorization
  #HS512 알고리즘을 사용할 것이기 때문에 512bit, 즉 64byte 이상의 secret key를 사용해야
한다.
  secret:
a2FyaW10b2thcmltdG9rYXJpbXRva2FyaW10b2thcmltdG9rYXJpbXRva2FyaW10b2thcmltdG9rYXJpb
XRva2FyaW10b2thcmltdG9rYXJpbXRva2FyaW10b2thcmltdG9rYXJpbXRva2FyaW10b2thcmltdG9rYX
JpbXRva2FyaW10b2thcmltdG9rYXJpbQ==
  token:
    #접근토큰시간: 5시간  #60*60*5*1000 =18000000
    access-expiration-time: 259200000
    #갱신토큰시간: 3일  #60*60*24*3*1000=259200000
```

```
    refresh-expiration-time: 259200000
springdoc:
  swagger-ui:
    groups-order: desc
    tags-sorter: alpha
    operations-sorter: method
    disable-swagger-default-url: true
    display-request-duration: true
    default-models-expand-depth: 2
    default-model-expand-depth: 2
  api-docs:
    path: /api-docs

  show-actuator: true
  default-consumes-media-type: application/json
  default-produces-media-type: application/json
  writer-with-default-pretty-printer: true
  model-and-view-allowed: true
  paths-to-match:
    - /api/**

pay:
  secret-key: ${PAY_SECRET_KEY}

blockchain:
  auth: ${BLOCKCHAIN_AUTH}
  chain-id: ${BLOCKCHAIN_CHAIN_ID}
```

나. application-dev.yml

```
spring:
  # 1) DB 연결 설정
  datasource:
    driver-class-name: org.mariadb.jdbc.Driver
    url: ${RDS_URL}?serverTimezone=Asia/Seoul&characterEncoding=UTF-8
    username: ${RDS_USERNAME}
    password: ${RDS_PASSWORD}

  # 2) jpa 설정
  jpa:
    hibernate:
      ddl-auto:  update
    properties:
      hibernate:
        format_sql: true # SQL이 로그창에 정갈하게 표시되도록
        show_sql: true # 로그창에 SQL이 뜨도록
        default_batch_fetch_size: 1000 #기본 batch size 설정


# 3) 포트번호 보이지 않도록 조치
server:
  servlet:
    encoding:
      force: 'true'
      enabled: 'true'
```

```
      charset: UTF-8
    context-path: /
  port: '8081' // Jenkins와 포트 번호가 겹치지 않게 하기 위한 포트 변경
```

3. 설정 파일

> Git Lab에서 Clone 후 배포하여 사용하는 과정 전체를 설명 드리겠습니다.

(1) EC2 접속

Pem.key가 있는 폴더에서 Linux 터미널을 오픈합니다.
해당 키를 이용하여 접속 합니다.

```
ssh -i (키 이름- 확장자까지) (우분투 주소)
ssh -i ./J10D210T.pem ubuntu@j10d210.p.ssafy.io
```

(2) Docker 설치

> CI/CD가 가능한 환경 조성을 위해, 여러 인프라 도구를 설치해야 합니다. 이를 위해서는 Docker가 필수적입니다.

가. 오래된 버전의 Docker 삭제 (기존 설치 시 실행)

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

나. Repository 설정

> CI/CD를 위한 여러 인프라 도구들은 도커 내에 이미지 형태로 저장됩니다. Docker Repository는 이러한 도구들의 이미지가 저장되는 공간입니다. Docket Repository는 GitHub의 Repository와 비슷한 개념이라 간주하면 편합니다.

```
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg lsb-release
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo 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/ubuntu $(lsb_release -cs) stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null
```

다. Docker Engine 설치

> Docker Engine은 Docker Components 와 서비스를 제공하는 **컨테이너**를 구축하고 실행하는 기본 핵심 소프트웨어 입니다.
> 개발자들이 흔히 Docker라고 말하는 것은 실제로 Docker-Engine이고, 해당 DockerEngine을 설치해야지, Container 형식의 Docker Engine 시스템을 사용할 수 있습니다.

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

라. 설치 완료 확인

```
docker --version
```

(3) Docker Compose 설치

가. 최신 버전 설정 후 Docker Compose 다운로드

> CD를 위해 Docker-Compose를 설치 합니다.

```
sudo apt install jq
VERSION=$(curl --silent
https://api.github.com/repos/docker/compose/releases/latest | jq .name -r)
DESTINATION=/var/lib/jenkins/workspace/WalkWalk
sudo curl -L
https://github.com/docker/compose/releases/download/${VERSION}/docker-
compose-$(uname -s)-$(uname -m) -o $DESTINATION
sudo chmod 755 $DESTINATION
```

> Compose의 내용은 다음과 같습니다. 저희팀은 환경 변수도 Compose에 설정해두어, Backend git push 시 기밀 정보가 GitLab에 올라가는 사항 자체를 방지하였습니다.

```
 services:
        nginx:
                build:
                        context: ./frontend/frontend
                ports:
                        - 3000:80
                depends_on:
                        - spring
        spring:
                build:
                        context: ./backend/backend
                environment:
                        - BLOCKCHAIN_AUTH=Basic
S0FTS044NFlBSDJaVTZCRTE5QOZTQ1o3OlhKclc1dTZ6MGRwb1dtWDdsMDlvVDlzcFJNZWtYRjc1bWp1W
FprMOI=
                        - BLOCKCHAIN_CHAIN_ID=1001
                        - GOOGLE_CLIENT_ID=661916935878-
j4hpr01ps70konotjd0kdpaoogt6dfai.apps.googleusercontent.com
                        -
GOOGLE_REDIRECT_URI=http://localhost:3000/oauth2/callback/google
                        - GOOGLE_SECRET=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
                        -
LOCAL_DATASOURCE_URL=jdbc:mariadb://localhost:3307/ssafy

                        - LOCAL_PASSWORD=ssafy
                        - LOCAL_USERNAME=root
                        - PAY_SECRET_KEY=DEVC4837C63F27FFE5B41932572CFCB74997BA93
                        - RDS_PASSWORD=ssafy1234
                        - RDS_URL=jdbc:mariadb://ssafyd210.c7iqguism6yd.ap-
northeast-2.rds.amazonaws.com:3307/ssafy
```

```
                        - RDS_USERNAME=admin
                        - REDIS_HOST=43.202.3.155
                        - REDIS_PASSWORD=ssafy1234
                        - REDIS_URL=http://j10d210.p.ssafy.io
                        - S3_ACCESS_KEY=AKIAQ3EGPF2TGWOZ4U7Z
                        - S3_SECRET_KEY=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
                        -
Google_TTS_CLOUD_API_KEY=AIzaSyBwAF_6T43ghHS6ytRUjJxr_qXyz6ye2hQ
                ports:
                        - 8081:8081
                depends_on:
                        - redis
        redis:
                image: redis
                ports:
                        - 6379:6379
```

(4) Nginx 설치

가. 설치

```
sudo apt install nginx
```

나. CertBot 설치

```
sudo add-apt-repository ppa:certbot/certbot # certbot을 위한 저장소 추가
sudo apt install python3-certbot-nginx      # certbot nginx 패키지 설치
```

다. CertBot 이용해 SSL 인증서 발급

```
sudo certbot certonly --nginx -d <도메인 주소>
```

▶ 인증서 발급 요청이 많아서 안 될 경우,

라. nginx 설정 파일 생성

> 경로는 다음과 같습니다.

```
sudo vim /etc/nginx/sites-available/nginx-settings.conf
```

> 설정파일은 다음과 같습니다.

```
server {
    location / {
        proxy_pass http://localhost:3000;
    }

    location /api {
        proxy_pass http://localhost:8081/api;
    }

    location /api/notifications/subscribe {
```

```
        proxy_pass http://localhost:8081/api/notifications/subscribe;
        proxy_set_header Connection '';
        proxy_set_header Cache-Control 'no-cache';
        proxy_set_header Content-Type text/event-stream;
        proxy_buffering off;
        proxy_http_version 1.1;
        chunked_transfer_encoding on;
        proxy_read_timeout 86400s;
    }

    // 소켓 설정
    location /ws-stomp {
        proxy_pass http://localhost:8081/ws-stomp;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";
        proxy_set_header Host $host;
    }

    listen 443 ssl http2; # Cert Config
    ssl_certificate /home/ubuntu/ssl/certificate.crt;
    ssl_certificate_key /home/ubuntu/ssl/private.key;
}

server {
    if ($host = j10d210.p.ssafy.io) {
        return 301 https://$host$request_uri;
    }

    listen 80;
    server_name j10d210.p.ssafy.io;
    return 404;
}
```

443 ssl 설정에 defualt 설정인 http1.1이 아닌 http2 로 설정되도록 하여 RestAPi 송수신 시에 처리 속도를 10~15% 올렸습니다.

마. sites-enabled에 심볼릭 링크 생성

```
sudo ln -s /etc/nginx/sites-available/nginx-settings.conf /etc/nginx/sites-enabled
```

▶ 심볼릭 링크를 다시 생성해야 해서 삭제해야할 경우

바. NGINX 실행 테스트

```
sudo nginx -t
```

사. NGINX 실행, 중지, 재시작 명령어

```
sudo systemctl start nginx
sudo systemctl restart nginx # 설정 파일 변경 후엔 restart를 해줘야 한다.
sudo systemctl stop nginx
sudo systemctl status nginx
```

(5) Docker에 각 팀 프로젝트 이미지로 올리기

가. React + Vite 프로젝트를 root 디렉토리에 올리기

> Frontend 프로젝트 root Directory에 DockerFile을 생성합니다.

```
FROM node:alpine as builder
WORKDIR /frontend
COPY package.json .
# RUN npm install
COPY ./ ./
RUN npm run build

# 3000번 포트 노출
EXPOSE 3000

# npm start 스크립트 실행
CMD ["npm", "run", "dev"]
```

> React + VITE 프로젝트의 루트 디렉토리에 nginx.conf란 이름으로 파일을 생성합니다.

```
server {
    listen 80;
    location / {
        root    /app/build;
        index   index.html;
        try_files $uri $uri/ /index.html;
    }
}
```

나. Spring Boot 프로젝트를 root 디렉토리에 올리기

> Springboot 프로젝트의 root 디렉토리에 DockerFile을 다음과 같이 생성합니다.

```
# 베이스 이미지
FROM openjdk:17

# 컨테이너 외부와 컨테이너 내부를 연결시켜 저장된 데이터의 수명과 데이터를 생성한 컨테이너의 수명을 분리
VOLUME /tmp

# 외부 호스트 8081 포트로 노출
EXPOSE 8081
```

```
# 이미지 생성 시 파일 복사
COPY build/libs/*.jar application.jar

# 컨테이너의 어플 지정
CMD ["java", "-jar", "application.jar"]
```

(6) Jenkins 설치

가. EC2 서버에 java 설치

```
sudo apt-get update
sudo apt-get install openjdk-17-jre
```

나. 젠킨스 설치

> 사전에 해야할 일: 먼저 설치한 젠킨스에 접근할 수 있도록 EC2 서버의 8080 Port를 열어야 합니다.

```
포트 여는 작업과 관련 명령어
ufw 적용 순서


제공되는 EC2의 ufw(우분투 방화벽)는 기본적으로 활성화(Enable) 되어 있고,
ssh 22번 포트만 접속 가능하게 되어 있습니다.

포트를 추가할 경우 6번부터 참고하시고,
처음부터 새로 세팅해 보실 경우에는 1번부터 참고하시기 바랍니다.


1. 처음 ufw 설정 시 실수로 ssh접속이 안되는 경우를 방지하기 위해
   ssh 터미널을 여유있게 2~3개 연결해 놓는다.

2. ufw 상태 확인
$ sudo ufw status
Status : inactive

3. 사용할 포트 허용하기 (ufw inactive 상태)
$ sudo ufw allow 22

3-1 등록한 포트 조회하기 (ufw inactive 상태)
$ sudo ufw show added
Added user rules (see 'ufw status' for running firewall):
ufw allow 22

4. ufw 활성화 하기
$ sudo ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)?
y

4.1 ufw 상태 및 등록된 rule 확인하기
$ sudo ufw status numbered
Status: active

     To                         Action      From
```

```
     --                         ------      ----
[ 1] 22                         ALLOW IN    Anywhere
[ 2] 22 (v6)                    ALLOW IN    Anywhere (v6)

5. 새로운 터미널을 띄워 ssh 접속해 본다.
C:\> ssh -i 팀.pem ubuntu@팀.p.ssafy.io

6. ufw 구동된 상태에서 80 포트 추가하기
$ sudo ufw allow 80

6-1. 80 포트 정상 등록되었는지 확인하기
$ sudo ufw status numbered
Status: active

     To                         Action      From
     --                         ------      ----
[ 1] 22                         ALLOW IN    Anywhere
[ 2] 80                         ALLOW IN    Anywhere
[ 3] 22 (v6)                    ALLOW IN    Anywhere (v6)
[ 4] 80 (v6)                    ALLOW IN    Anywhere (v6)

6-2. allow 명령을 수행하면 자동으로 ufw에 반영되어 접속이 가능하다.

7. 등록한 80 포트 삭제 하기
$ sudo ufw status numbered
Status: active

     To                         Action      From
     --                         ------      ----
[ 1] 22                         ALLOW IN    Anywhere
[ 2] 80                         ALLOW IN    Anywhere
[ 3] 22 (v6)                    ALLOW IN    Anywhere (v6)
[ 4] 80 (v6)                    ALLOW IN    Anywhere (v6)

7-1. 삭제할 80 포트의 [번호]를 지정하여 삭제하기
      번호 하나씩 지정하여 삭제한다.
$ sudo ufw delete 4
$ sudo ufw delete 2
$ sudo ufw status numbered  (제대로 삭제했는지 조회해보기)
Status: active

     To                         Action      From
     --                         ------      ----
[ 1] 22                         ALLOW IN    Anywhere
[ 2] 22 (v6)                    ALLOW IN    Anywhere (v6)

7-2 (중요) 삭제한 정책은 반드시 enable을 수행해야 적용된다.
$ sudo ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)?
y입력


기타
- ufw 끄기
$ sudo ufw disable
```

> 다음과 같이 Jenkins Debian Package를 설치합니다.

```
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
    https://pkg.jenkins.io/debian/jenkins.io-2023.key

echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
    https://pkg.jenkins.io/debian binary/ | sudo tee \
    /etc/apt/sources.list.d/jenkins.list > /dev/null

sudo apt-get update

sudo apt-get install fontconfig openjdk-17-jre

sudo apt-get install jenkins
```

Jenkins Debian Package 참고 사이트

> 젠킨스 설치를 확인

```
sudo systemctl status jenkins
```

다. Jenkins 실행

> a. http://도메인주소:8080 에 접속합니다.

▶ 접속이 안된다면?

라. Jenkins 초기 비밀번호 설정

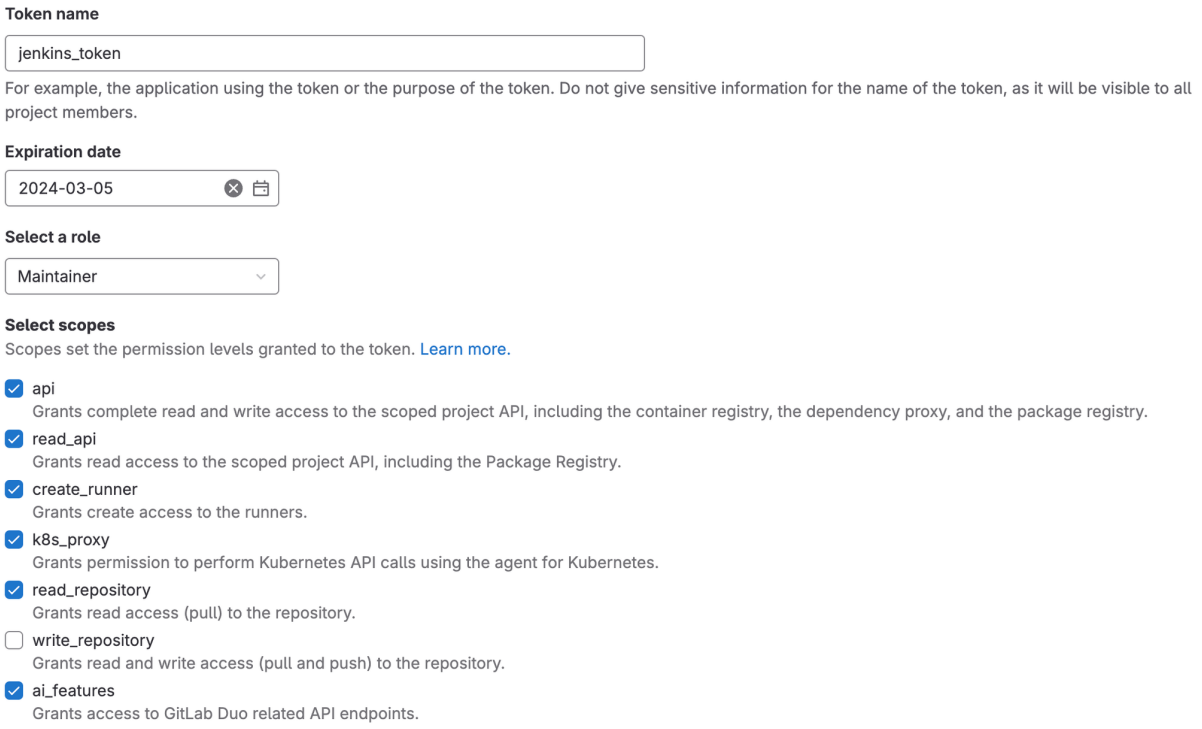
```
sudo cat ~/my-jenkins/jenkins_home/secrets/initialAdminPassword
```

마. 젠킨스 플러그인 설치

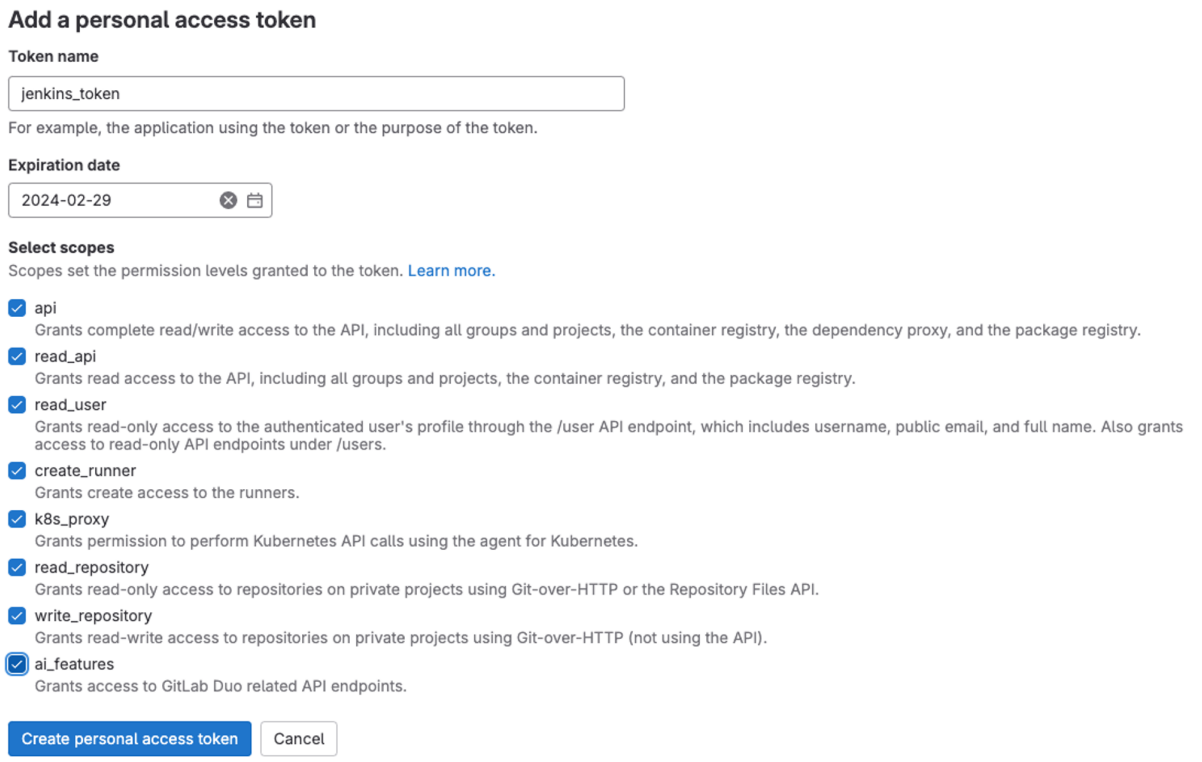
- GitLab
- Publish Over SSH
- Mattermost Notification
- SSH Agent
- Docker
- Jenkins 관리 -> Tools -> Gradle, NodeJs Intsll (위에 기입한 프로젝트 버전에 맞게 설치한다.)
  (WALK_WALK 프로젝트: Gradle 8.5, NodeJs 20.11.0, Git)

설치

바. Jenkins GitLab 연결하고 웹 훅 설정하기



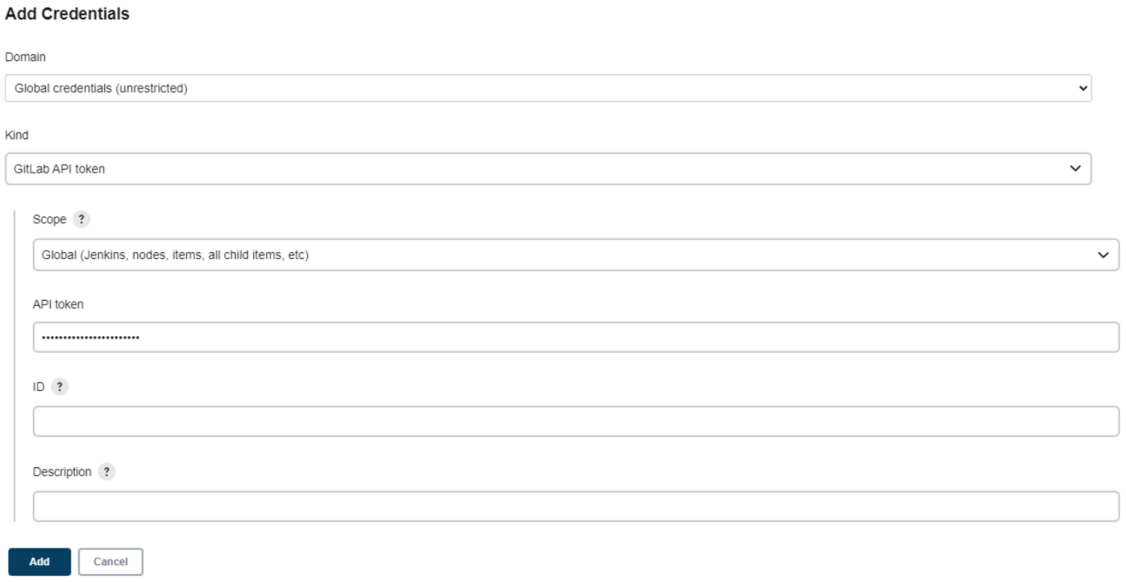
- GIt Lab -> 프로필 -> Preferences -> Access Tokens -> Add new token



사. 젠킨스 설정

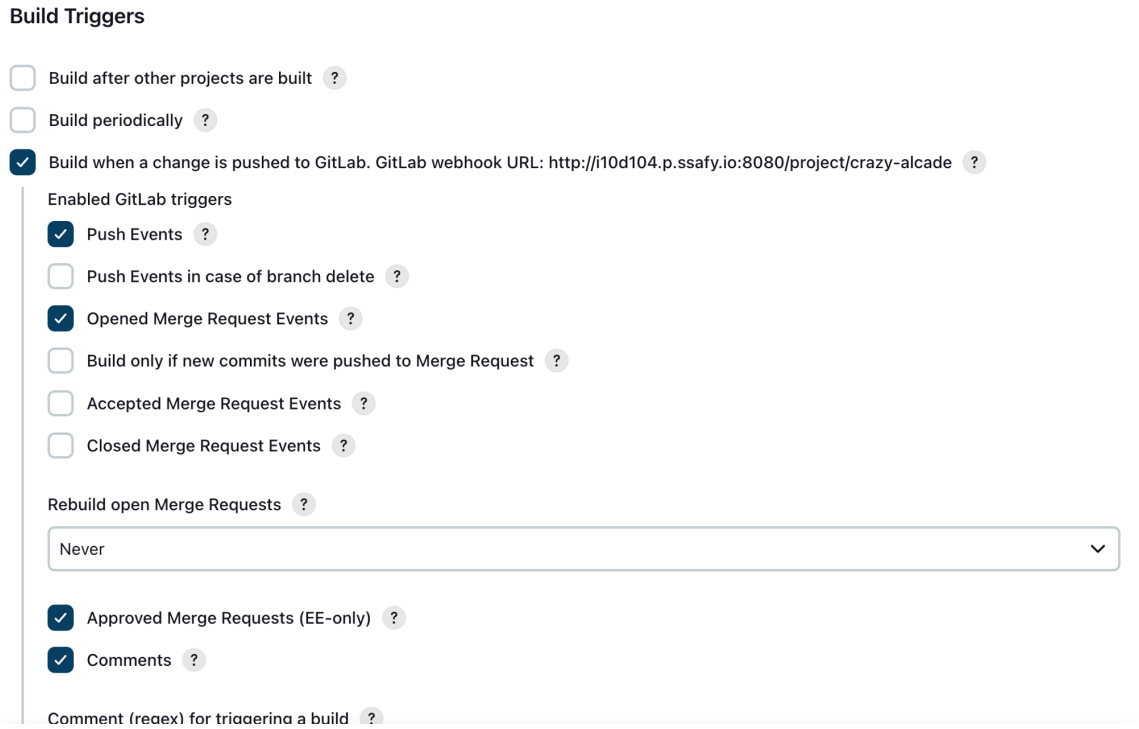
- jenkins 관리 - 시스템 설정 - GitLab

- Add Credentials



바. 젠킨스 Webhook 설정

- 젠킨스 - 새로운 Item - PipeLine
- Configure - Build Triggers
    - Build when a change is pushed to GitLab

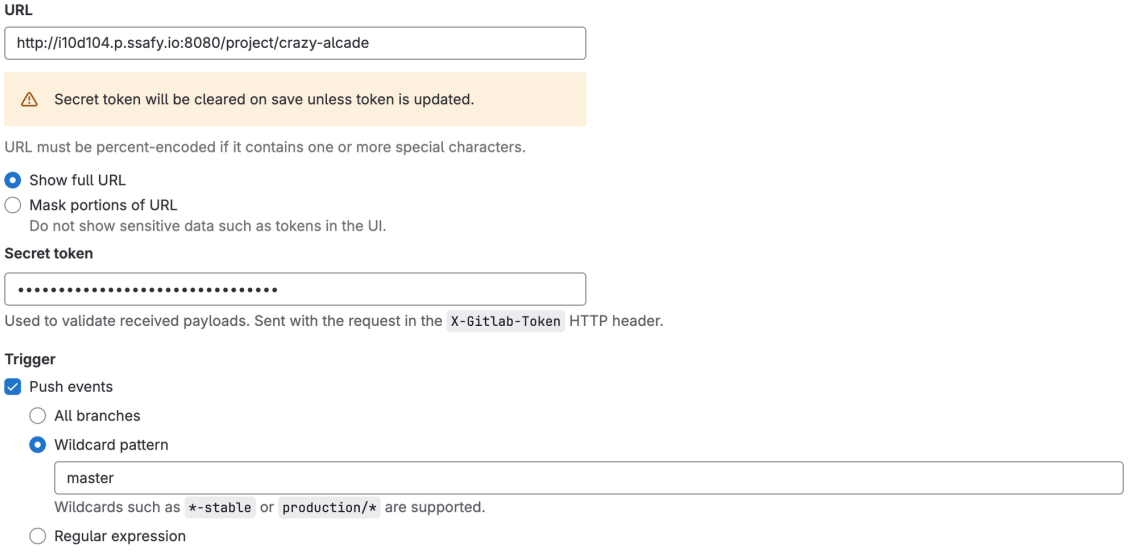


여기 GitLab Webhook URL 기억해두기

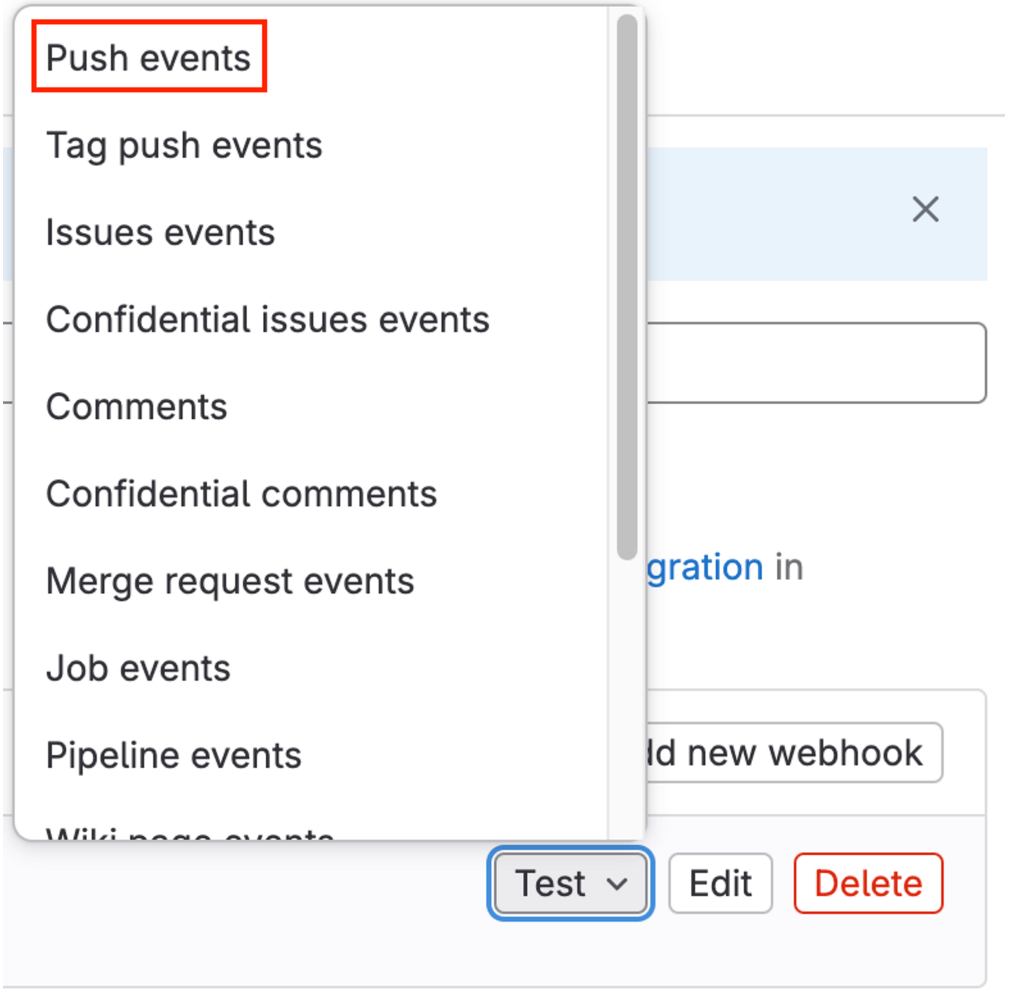
- 고급 -> Secret Token 생성

사. GitLab WebHook 설정

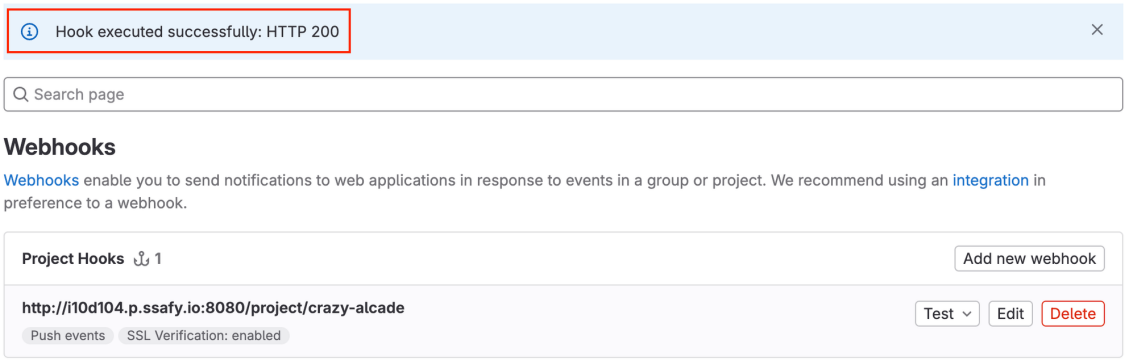
- Repository - Settings - Webhooks



- Test



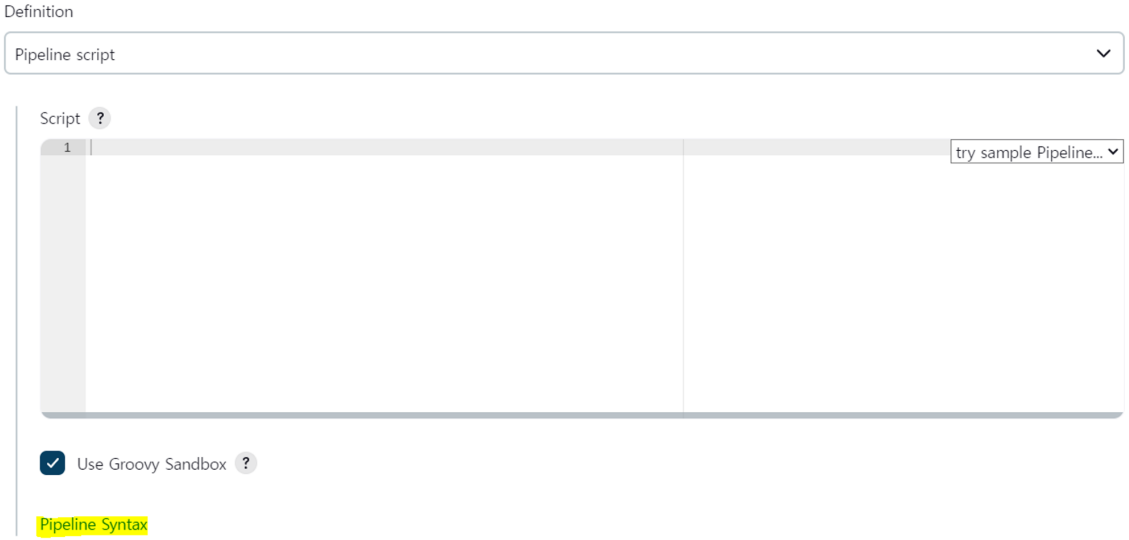
Push events를 발생 시켜보겠습니다.



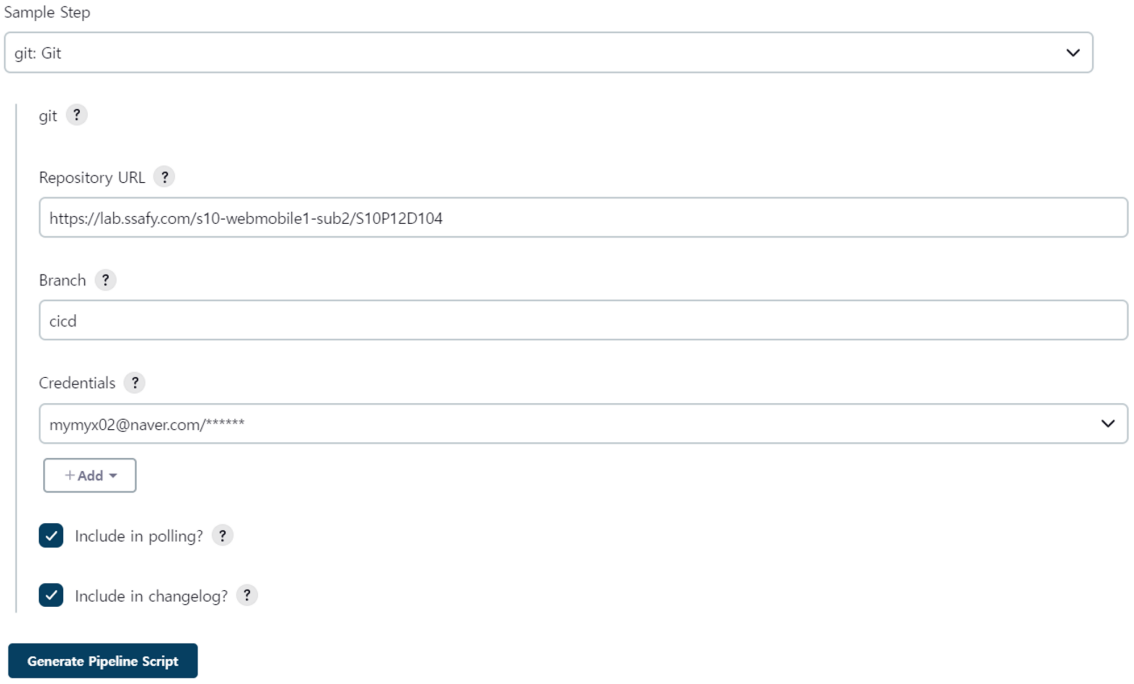
(7) 파이프라인 스크립트 작성

가. 기본 설정

- Jenkins에 Key를 추가합니다.



- Pipeline Syntax



- Sample Step: git
- Repository URL 입력
- Branch 입력
- Add
  - Kind: Username with password
  - Username: 깃랩 아이디 (이메일)
  - Password: 깃랩 개인 토큰
- Generate Pipeline Script
- 만들어진 credentialsId 복사해두기 (나는 직접 지정해줬다.)

나. 스크립트 작성

```
pipeline {
    agent any

    tools {
        gradle "gradle"
        nodejs "nodejs"
    }


    stages {
        stage('clone_frontend'){
            steps{
                dir('frontend'){
                    git branch: 'frontend', credentialsId: 'b648de27-d697-4f12-925c-f84a8fb878ab', url: 'https://lab.ssafy.com/s10-blockchain-contract-sub2/S10P22D210'
                }
            }
        }
        stage('front_build'){
            steps{
                dir('frontend'){
                    dir('frontend'){
                        sh 'npm install'
                        sh 'npm run build'
                    }
                }
            }
        }
        stage('clone_backend'){
            steps{
                dir('backend'){
                    git branch: 'backend', credentialsId: 'b648de27-d697-4f12-925c-f84a8fb878ab', url: 'https://lab.ssafy.com/s10-blockchain-contract-sub2/S10P22D210'
                }
            }
        }
        stage('back_build'){
            steps{
                dir('backend'){
                    dir('backend'){
                        sh 'gradle clean build'
                    }
                }
            }
        }
        stage('deploy'){
            steps{
                sh 'docker-compose up -d --build'
                mattermostSend color: '#c983f8', message: "두근두근... >_< 배포성공!! \n (${env.JOB_NAME}) #(${env.BUILD_NUMBER}) (<${env.BUILD_URL}|Open>) \n See the (<${env.BUILD_URL}console|console>)"
```

```
            }
        }
        stage('docker-clean'){
            steps{
                sh 'docker container prune -f'
                sh 'docker image prune -f'
            }
        }
    }
}
```