포팅 메뉴얼

1. 개발 환경

1.1. Frontend

- Node.js 22.5.1(LTS)
- React 18.3.1
  - Redux 9.1.2
  - Reduxjs/Toolkit 2.2.6
  - Router 6.25.1
- axios 1.7.2
- animejs 3.2.2
- socket-io: 4.7.5
- Openvidu Browser 2.30.1
- jwt-decode 4.0.0
- styled-components 6.1.12
- websocket 1.0.35

1.2. Backend

- Spring Boot
  - Spring Web

    - Spring Reactive Web
    - Spring Security
    - OAuth2 Client
- DB
    - JDBC API
    - Spring Data JPA
    - H2 Database
    - MySQL Driver
    - Spring Data Redis
    - Redisson Spring boot starter
- Websocket
    - Java Mail Sender
- SMTP
    - WebSocket
    - Spring Boot Actuator
- openvidu-java-client 2.30.0
- Utill
    - Lombok
    - Validation

1.3. Server

- Ubuntu 20.04 LTS
- Nginx 1.18.0
- Docker 27.0.3
- Docker Compose 2.28.1

- OpenVidu 2.30.0
- Jenkins 2.45.2.3

1.4. DB

- H2 (Develop)
- MySQL 8.0 (Deploy)
- Redis 6.2.14

1.5. 형상 / 이슈 관리

- Jira
- GitLab

2. EC2 세팅

2.1. Docker Engine 설치

ref) https://docs.docker.com/engine/install/ubuntu/#install-using-the-repository

```
# 기존에 설치된 도커 삭제
sudo apt-get purge docker-ce docker-ce-cli containerd.io docker-

# Docker 설치 ( 공식문서 참고 )
# Repository 추가
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
```

```
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/key
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update

# Install Docker Package (include Docker compose)
sudo apt-get install docker-ce docker-ce-cli containerd.io docke

# After Installation
sudo groupadd docker
sudo usermod -aG docker $USER
newgrp docker
```

2.2. OpenVidu 설치 (On Premise)

ref) https://docs.openvidu.io/en/stable/deployment/ce/on-premises/

개발 당시 OpenVidu v3이 Beta 버전이 출시되어 있었으나, 안정성을 위해 2.30.0 버전으로 개발 및 배포를 진행함.

⚠ Vidu 포트 변경전 NGINX 가 실행 중일 경우, 종료 후 작업 진행

→ 포트 충돌로 인해 정상적으로 인증서 발급이 안될 수 있음.

⚠ openvidu와 관련된 docker image / container가 없어야 함.

→ 버전이 맞지 않을 경우, 에러가 발생할 수 있음.

```
# 1. OpenVidu 서버 생성
sudo su
```

```
cd /opt
curl https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/install_
cd openvidu

# 2. Vidu Container SSL 인증서 발급
# 2-1. .env파일 수정
nano .env
### .env
DOMAIN_OR_PUBLIC_IP=<Project Domain>
OPENVIDU_SECRET=<OpenVidu Password>
CERTIFICATE=letsencrypt
LETSENCRYPT_EMAIL=<INFRA_MANAGER_EMAIL>
# 해당 파일 저장 후 OpenVidu 실행
./openvidu start
# container 모두 실행된 후 -> openvidu media server SSL 적용완료
./openvidu stop

# 2-2. .env 파일 수정
HTTP_PORT=<HTTP_PORT>
HTTPS_PORT=<HTTPS_PORT>

# 3. OpenVidu 실행 ( 적용 완료 )
./openvidu start
```

2.3. Nginx Reverse Proxy 설정 + SSL 인증서 발급

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

# 2. Let's Encrypt 설치 및 SSL 발급
sudo apt-get install letsencrypt
sudo systemctl stop nginx
sudo letsencrypt certonly --standalone -d ${프로젝트도메인}
```

```
# 3. Nginx 설정파일 생성
cd /etc/nginx/sites-available
sudo rm -rf default // 혹시 모를 오류를 대비해 기존 설정파일 삭제
sudo nano configure

######################
server{

    if ($host = i11e106.p.ssafy.io) {
        return 301 https://$host$request_uri;
    }
    listen 80;

    server_name i11e106.p.ssafy.io;
    return 404;
}

# map $http_upgrade $connection_upgrade {
#   default upgrade;
#   '' close;
# }

server{
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name i11e106.p.ssafy.io;

    ssl_certificate /etc/letsencrypt/live/i11e106.p.ssafy.io/ful
    ssl_certificate_key /etc/letsencrypt/live/i11e106.p.ssafy.io

    access_log /var/log/nginx/jenkins.access.log;
    error_log /var/log/nginx/jenkins.error.log;

    ignore_invalid_headers off;

    location /jenkins/ {
```

```
        proxy_pass http://localhost:8081/jenkins/;
        proxy_redirect off;

        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_
        proxy_set_header Host $host:$server_port;
        proxy_set_header X-Forwarded-Proto http;
        proxy_set_header X-Forwarded-Port "443";
        proxy_set_header X-Forwarded-Host $http_host;
    }

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

    location /api/ {
        proxy_pass http://localhost:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    location /ws {
        proxy_pass http://localhost:8080/ws;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_
        proxy_set_header Host $host;

        #WebSocket Support
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
```

```
}
######################
```

2.3. EC2 포트 정리

포트번호	내용
22	SSH
80	HTTP (HTTPS 로 redirect)
443	HTTPS
3000	Frontend Deploy (Docker)
8080	Backend Deploy (Docker)
3478	OpenVidu (TURN/STUN)
8443	OpenVidu (Media Server)
6379	DB - MySQL (Docker)
3306	DB - Redis (Docker)
8081	Jenkins

2.4. 방화벽(UFW) 설정

```
# 1. 위의 포트 정리에 해당하는 포트들 개방
# !!!IMPORTANT 22(SSH), 80(HTTP), 443(HTTPS) 포트 절대 건들지 마세요
# 단일 포트 열기
sudo ufw allow <포트번호>/<tcp/udp>
# 범위 포트 열기
sudo ufw allow <포트시작-포트종료>/<tcp/udp>

# 2. Firewall 활성화 / 상태 확인 / 적용
sudo ufw enable
sudo ufw status verbose/numbered
sudo ufw reload
```

```
# 3. 포트 차단
sudo ufw deny <포트번호>/<tcp/udp>
```

3. Deploy

3.1. Frontend

```
# 1. Home Directory로 이동
cd ~

# 2. Git Clone
git clone https://lab.ssafy.com/<git주소>

# 3. Move to Working Directory
cd <PRJ_FOLDER>/frontend

# 4. nginx.conf 작성
nano nginx.conf
######################
server {
        listen 80;
        location / {
                root /app/build;
                index index.html;
            try_files $uri $uri/ /index.html;
        }
}
######################

# 5. Dockerfile 생성
nano Dockerfile
######################
FROM nginx
RUN mkdir /app
```

```
WORKDIR /app
RUN mkdir ./build
ADD ./dist ./build
RUN rm /etc/nginx/conf.d/default.conf
COPY ./nginx.conf /etc/nginx/conf.d
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
######################

# 6. npm 모듈 설치
npm install

# 7. build 파일 생성
npm run build

# 8. 도커이미지 생성
docker build -t emissary-fe:0.1 .
docker rm - rf emissary-fe # 작동중이던 container 삭제
docker image prune # 사용중이지 않은 image 삭제

# 9. 도커이미지 실행 ( 컨테이너 생성 )
docker run --name emissary-fe -d -p 3000:3000 emissary-fe:0.1
```

3.2. Backend

```
# 1. Home Directory로 이동
cd ~

# 2. Git Clone
git clone https://lab.ssafy.com/<git주소>

# 3. Move to Working Directory
cd <PRJ_FOLDER>/backend

# 4. Dockerfile 작성
```

```
nano Dockerfile
######################
FROM openjdk:17-alpine
VOLUME /tmp
ARG JAR_FILE=build/libs/emissary_backend-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "/app.jar"]
ENV TZ=Asia/Seoul
RUN apk add --no-cache tzdata && \
    cp /usr/share/zoneinfo/Asia/Seoul /etc/localtime && \
    echo "Asia/Seoul" > /etc/timezone && \
    apk del tzdata
######################

# 5. Gradle 빌드파일 생성
## 권한설정
sudo chmode +x +R .
./gradlew clean build -x test

# 6. Docker image 생성
docker build -t emissary-be:0.1 .
docker rm - rf emissary-be # 작동중이던 container 삭제
docker image prune # 사용중이지 않은 image 삭제

# 9. 도커이미지 실행 ( 컨테이너 생성 )
docker run --name emissary-be -d -p 8080:8080 emissary-be:0.1
```

3.3. DB

DB 배포를 위해 Docker Compose 작성

```
version: "3.8"
services:
```

```
  mysql:
    image: mysql:8.0
    container_name: emissary_mysql
    # networks:
    #   - default
    #   - docker-network
    restart: on-failure
    ports:
      - ${MYSQL_BINDING_PORT}:${MYSQL_PORT}
    env_file:
      ./.env
    volumes:
      - ./db/mysql/data:/var/lib/mysql
      - ./db/mysql/init:/docker-entrypoint-initdb.d
    platform: linux/x86_64

  redis:
    image: redis:6.2.14-alpine
    container_name: emissary_redis
    # networks:
    #   - default
    #   - docker-network
    ports:
      - ${REDIS_BINDING_PORT}:${REDIS_PORT}
    command: redis-server
    volumes:
      - ./db/redis/data:/data
      - ./db/redis.conf:/usr/local/etc/redis/redis.conf

networks:
  default:
    name: docker-network
    external: true
```

4. CI/CD 환경 구축

Jenkins를 활용하여 CI/CD 환경을 구축하였습니다. 또한 GitLab WebHooks 설정을 통해 master 브랜치로 push 시 자동으로 빌드와 배포를 진행하도록 설정하였습니다.

4.1. Jenkins 도커 이미지 + 컨테이너 생성

Jenkins Docker Container 생성을 위한 `Dockerfile` 생성 (Jenkins 컨테이너 위에 Docker 설치)

```
# 1. Jenkins 폴더 생성
cd ~
mkdir jenkins
cd jenkins
nano Dockerfile
```

Dockerfile

```
FROM openjdk:17-alpine
VOLUME /tmp
ARG JAR_FILE=build/libs/emissary_backend-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "/app.jar"]
ENV TZ=Asia/Seoul
RUN apk add --no-cache tzdata && \
    cp /usr/share/zoneinfo/Asia/Seoul /etc/localtime && \
    echo "Asia/Seoul" > /etc/timezone && \
    apk del tzdata
```

docker-compose.yml

```
version: '3.8'
services:
  jenkins:
```

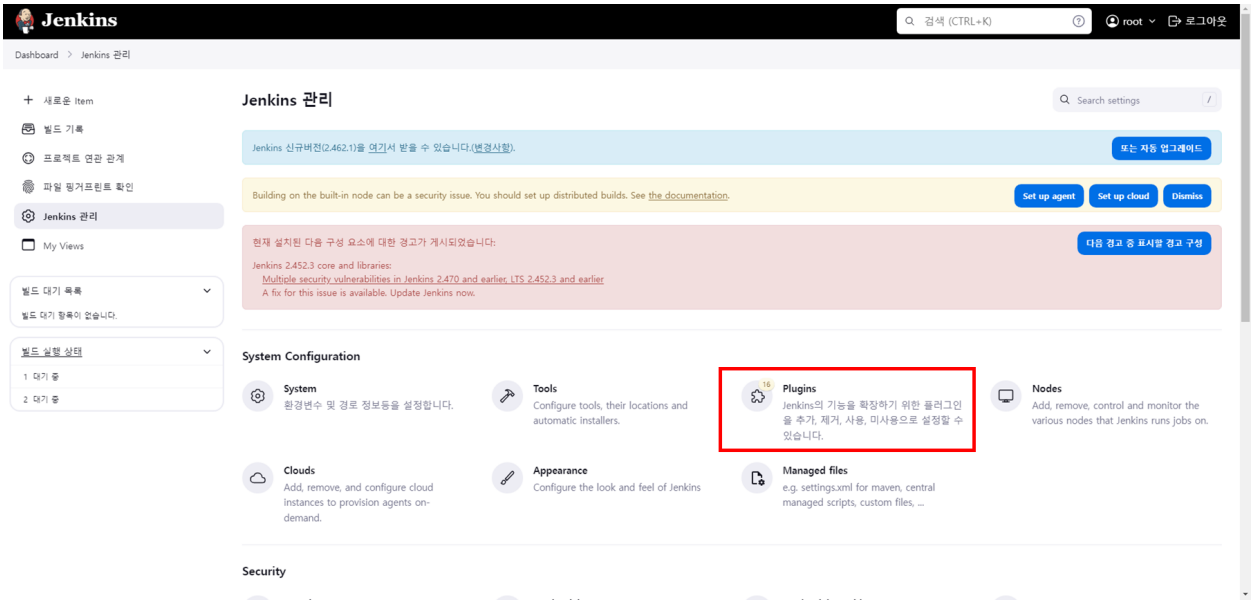
```
    build:
      context: .
    container_name: jenkins
    restart: on-failure
    environment:
      - JENKINS_OPTS="--prefix=/jenkins"
    user: root
    privileged: true
    ports:
      - 8081:8080
    volumes:
      - ./jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
```

jenkins도 Reverse Proxy 적용을 위해 `JENKINS_OPTS="--prefix=/jenkins"` 를 추가하였습니다.

4.2. Jenkins 설정

4.2.1. 플러그인 설치

Jenkins Home → Jenkins 관리 → Plugins → “Available plugins”클릭 → 플러그인 검색 및 설치

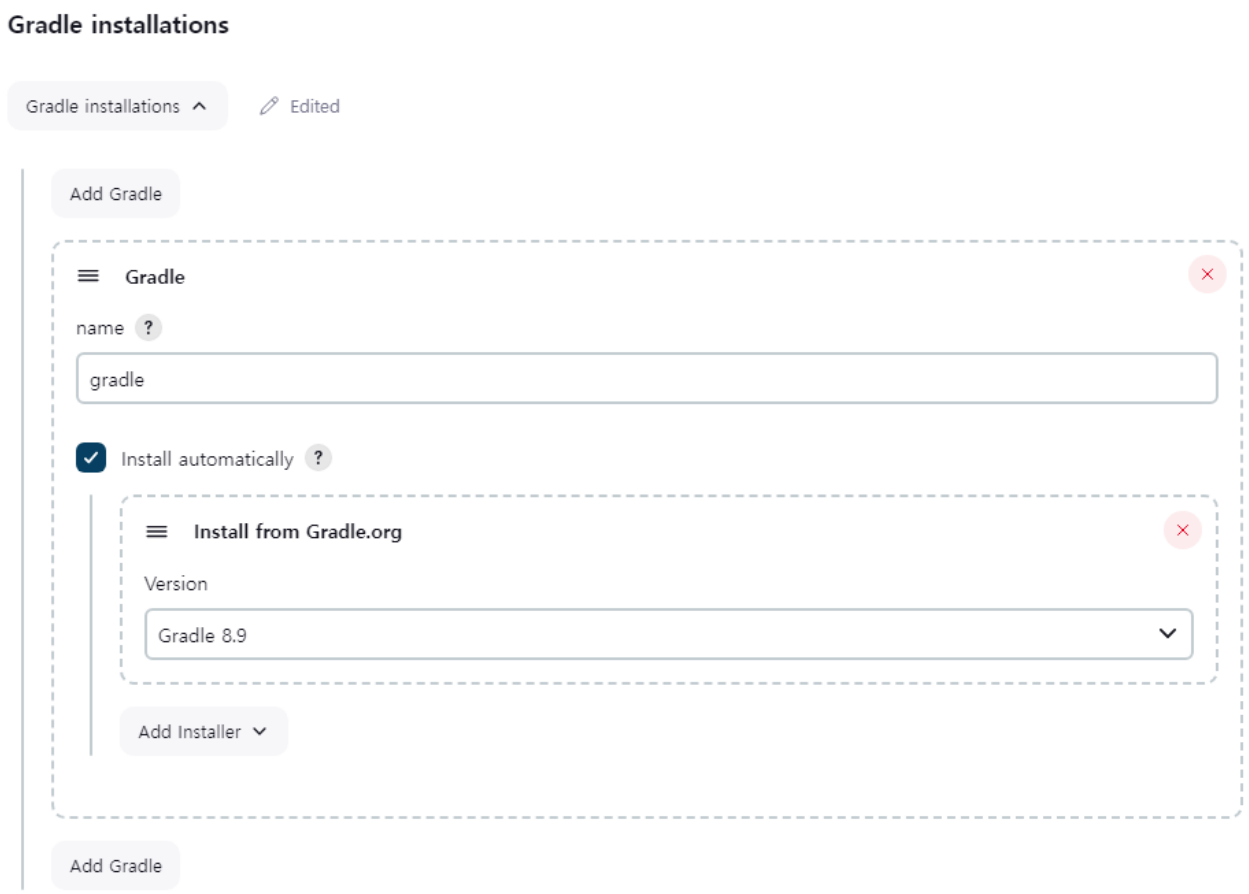


플러그인 목록

- Gradle
- GitLab Plugin
- GitHub API
- NodeJS Plugin

4.2.2 개발 도구 설정

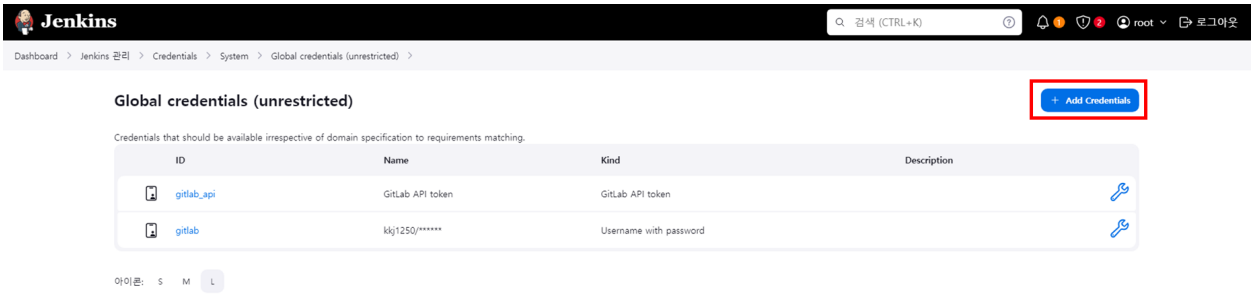
- Gradle installations



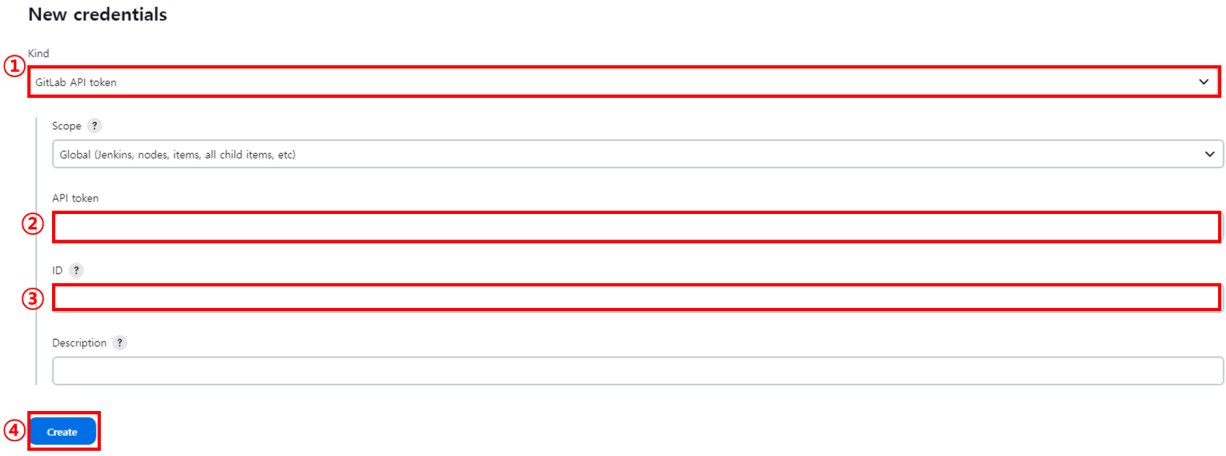
- NodeJS installations

4.2.3. GitLab Credentials 설정

Jenkins 홈 → Jenkins관리 → Credentials → System → Global credentials (unrestricted) 접속



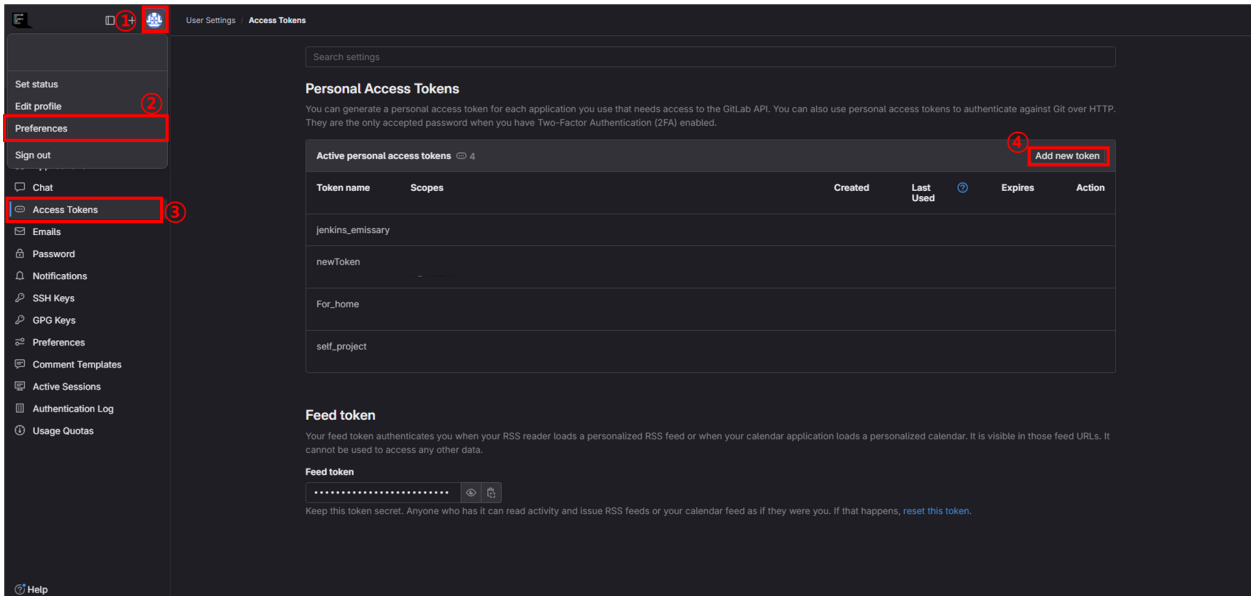
`+ Add Credentials` 버튼 클릭



1. Kind - Github API Token 선택
2. API token `4.3 Gitlab Credential` 참고
3. ID 입력 (추후 Pipeline 작성 시 변수명으로 사용됨)

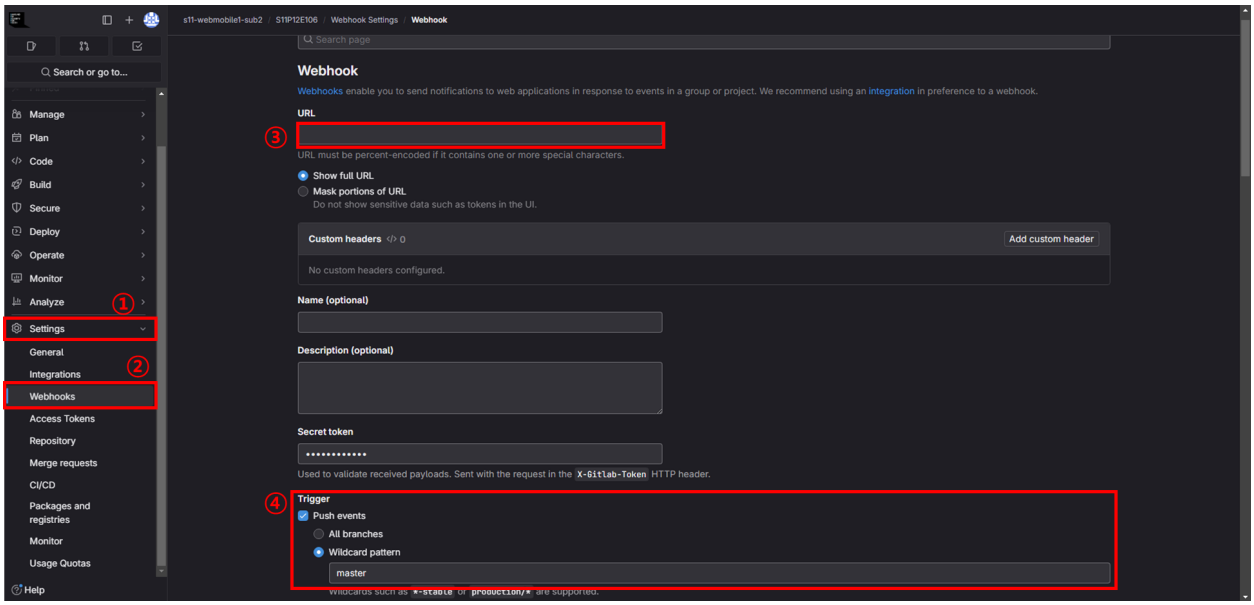
4.2.4. GitLab 설정 (Access Token 발급, WebHook 설정)

- **Access Token 발급**



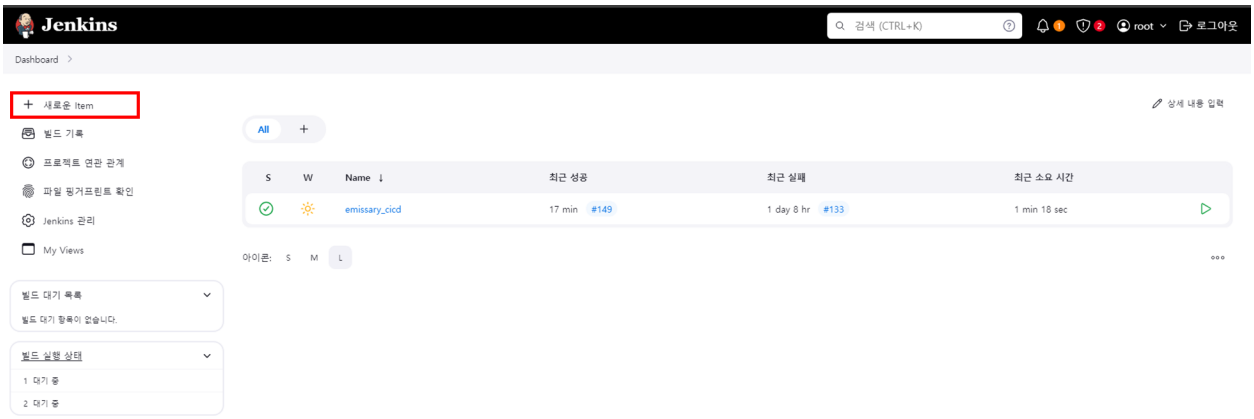
Gitlab Home → User Profile → Preference → AccessToken → Add New Token 진입

- **WebHook 설정**

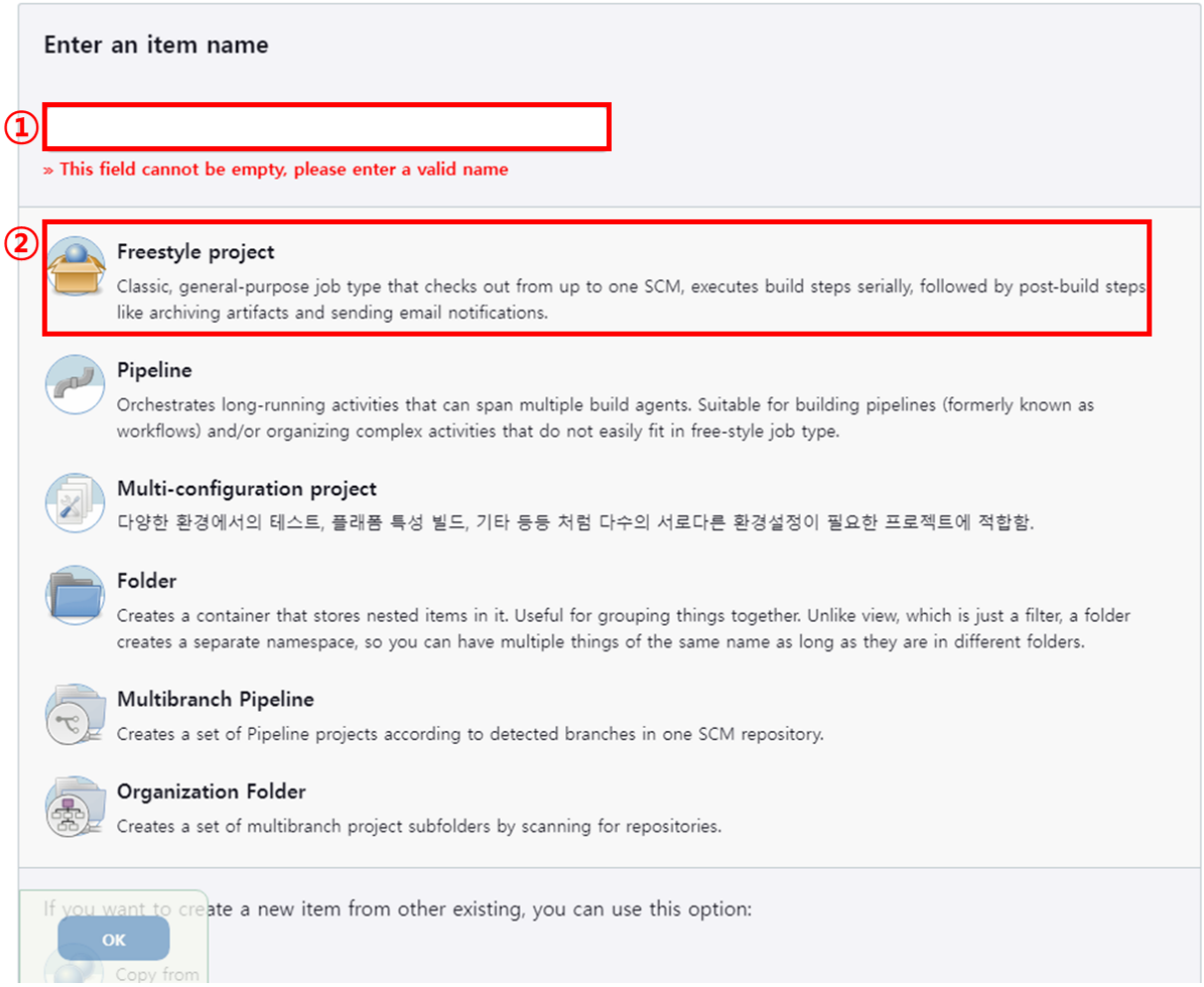


GitLab Project page → Settings → Webhooks → create webhook → ③ URL에 프로젝트 도메인 입력 → Trigger Push events → WildCardPattern(master)

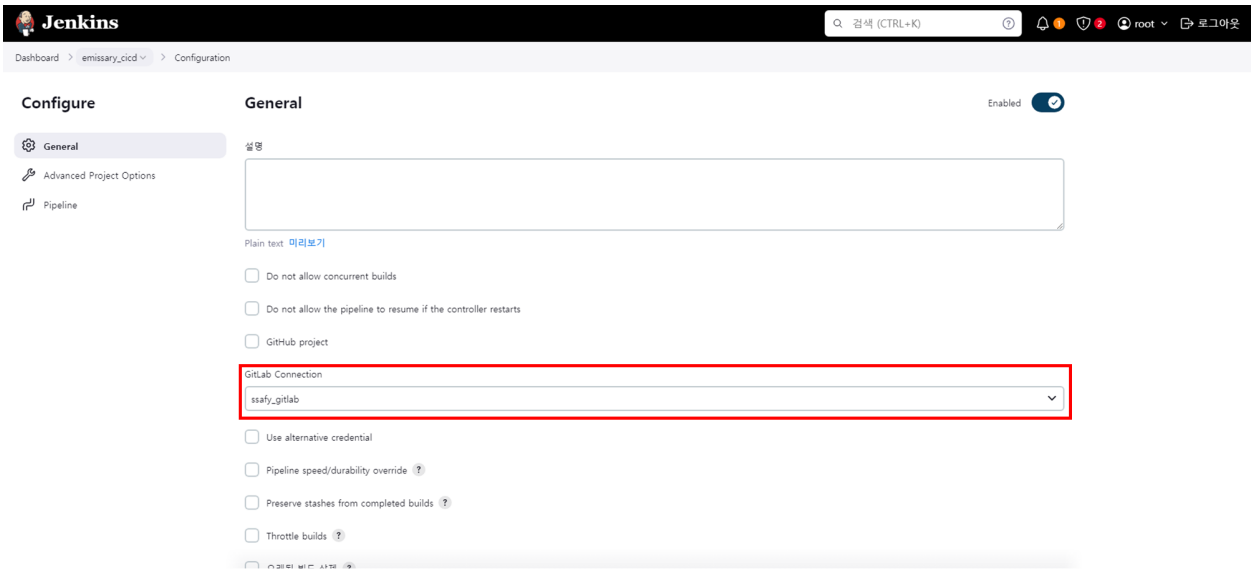
4.2.5. Jenkins Item 생성



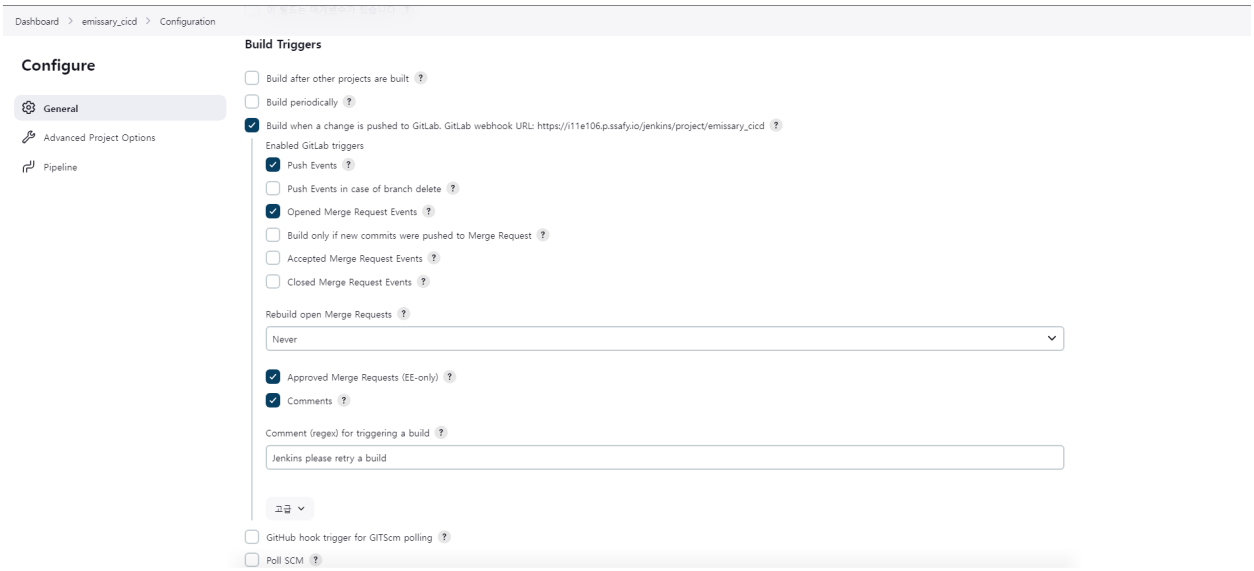
Jenkins 홈에서 `새로운 Item` 클릭



1. Pipeline 이름 입력
2. Freestyle Project (or Pipline) 둘 다 가능하지만 본 문서에서는 Freestyle Project로 소개함.



GitLab Connections → 4.2.2에서 설정한 ID 입력



Build Trigger 위와 같이 설정

Pipeline

```
pipeline {
    agent any

    tools{
```

```
        gradle 'gradle'
        nodejs 'nodeJS'
    }

    stages {
        stage('Git Clone'){
            steps{
                git branch: 'master',
                credentialsId: 'gitlab',
                url: 'https://lab.ssafy.com/s11-webmobile1-sub2/
            }
        }

        stage('Build - PRE SETTING'){
            steps{
                sh "chmod +x -R ${env.WORKSPACE}"
            }
        }

        stage('Build - FE'){
            steps{
                dir("${env.WORKSPACE}/frontend"){
                    nodejs(nodeJSInstallationName: 'nodeJS'){
                        sh 'rm -rf ./node_modules ./package-locl
                        sh 'npm install && npm run build'
                    }
                }
            }
        }

        stage('Build - BE'){
            steps{
                dir("${env.WORKSPACE}/backend"){
                    sh './gradlew clean build --exclude-task tes
                }
            }
```

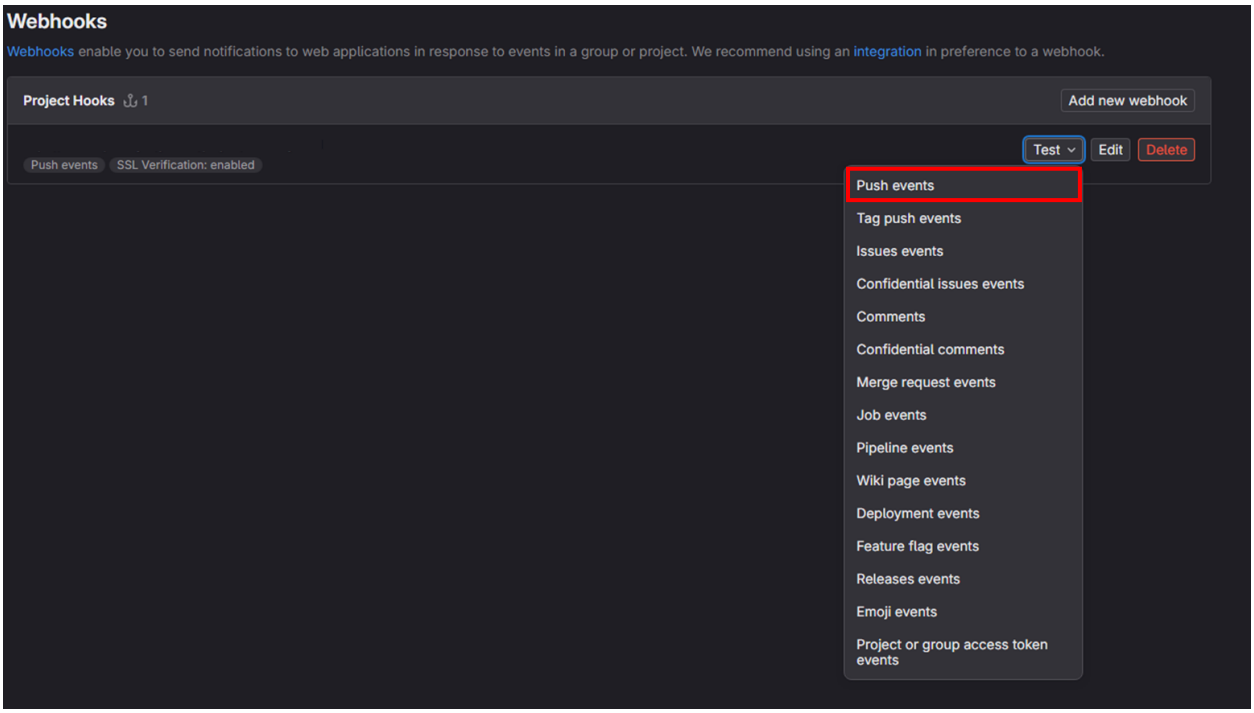
```
        }

        stage('Build - Docker'){
            steps{
                dir("${env.WORKSPACE}/deploy"){
                    sh 'docker compose build --no-cache'
                }
            }
        }

        stage('Deploy'){
            steps{
                dir("${env.WORKSPACE}/deploy"){
                    sh '''
                    docker compose up -d
                    docker image prune -f
                    '''
                }
            }
        }
    }
}
```

4.2.6. 빌드 및 배포

- Option 1. 상기 설정한 WebHook Branch 에 Push
- Option 2. GitLab Webhook에서 Test로 hook 전송



- Option 3. Jenkins 홈 화면 → Jenkins Item 클릭 → “지금 빌드” 클릭