B202 Wine Dining 포팅 메뉴얼

> **SSAFY 12기 특화프로젝트 대전 2반 2팀**

0. 목차

1. 개발 환경

1. 프로젝트 기술 환경 스택

- Frontend
  - Visual Studio Code 1.97.2
  - HTML5, CSS3, Javascript(ES6)
  - React 18.3.0
  - Vite 6.0.5
  - Tailwind CSS 4.0.0
  - Nodejs 20
  - Firebase 11.3.0
- Backend
  - Intellij 2025.1.6
  - Oracle Open JDK 17
  - JWT
  - SpringBoot 3.3.7
    - JAVA Spring Data JPA 3.1
    - Spring Security 6.4.2
  - Gradle 8.12.1
- CI/CD
  - AWS EC2
    - NGINX 1.27.4
    - Ubuntu 24.04.1 LTS
    - Docker 27.5.1
    - Jenkins 2.496
- Database
  - MySQL 8.0.41
  - Redis 7.4.2
  - S3

2. 환경변수 설정

Frontend

- .env

```
# 로컬용
VITE_API_BASEURL=
```

```
# 배포용
VITE_API_BASEURL=
```

Backend

- application.yml

```
spring:
  profiles:
    active: prod
```

- application-prod.yml

```
spring:
  datasource:
    url: jdbc:postgresql://<DB_HOST>:5432/<DB_NAME>?serverTimezone=Asia%2FSeoul&characterEncoding=UTF-8
    username: <DB_USERNAME>
    password: <DB_PASSWORD>
    driver-class-name: org.postgresql.Driver

  sql:
    init:
      mode: always

  jpa:
    database-platform: org.hibernate.dialect.PostgreSQLDialect
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        default_schema: public
        format_sql: true
        session.events.log: true
        session_factory:
          statement_inspector: com.ssafy.winedining.global.common.aop.SqlStatementInspector
    defer-datasource-initialization: true

  servlet:
    multipart:
      enabled: true
      max-file-size: 50MB
      max-request-size: 50MB

  data:
    redis:
      host: <REDIS_HOST>
      port: 6379
      password: <REDIS_PASSWORD>

recommendation:
  api:
    url: http://<RECOMMENDATION_API_HOST>:8000/api/v1/recommend/

logging:
  level:
```

```
    root: info
    org.hibernate.SQL: debug
    org.hibernate.type.descriptor.sql.BasicBinder: trace
    org.springframework.security: debug
    com.ssafy.winedining: debug
```

- application-secret.yml

```
spring:
  security:
    oauth2:
      client:
        registration:
          kakao:
            client-id: <KAKAO_CLIENT_ID>
            client-secret: <KAKAO_CLIENT_SECRET>
            redirect-uri: "https://{baseHost}/login/oauth2/code/{registrationId}"
            authorization-grant-type: authorization_code
            client-authentication-method: client_secret_post
            client-name: kakao
            scope:
              - profile_nickname
              - account_email

          google:
            client-id: <GOOGLE_CLIENT_ID>
            client-secret: <GOOGLE_CLIENT_SECRET>
            redirect-uri: "https://{baseHost}/login/oauth2/code/{registrationId}"
            authorization-grant-type: authorization_code
            client-authentication-method: client_secret_post
            client-name: google
            scope:
              - profile
              - email

        provider:
          kakao:
            authorization-uri: https://kauth.kakao.com/oauth/authorize
            token-uri: https://kauth.kakao.com/oauth/token
            user-info-uri: https://kapi.kakao.com/v2/user/me
            user-name-attribute: id

jwt:
  secret: <JWT_SECRET_KEY>
  access-token-expiration: 1209600000  # 14일
  refresh-token-expiration: 1209600000  # 14일

frontend:
  url: https://<FRONTEND_DOMAIN>/home

server:
  domain: <SERVER_DOMAIN>

cloud:
  aws:
    s3:
```

```
    bucket: <AWS_S3_BUCKET>
  stack.auto: false
  region.static: ap-northeast-2
  credentials:
    accessKey: <AWS_ACCESS_KEY>
    secretKey: <AWS_SECRET_KEY>

openai:
  api:
  key: <OPENAI_API_KEY>
  url: https://api.openai.com/v1/chat/completions
  model: gpt-3.5-turbo
```

FastAPI

- .env

```
POSTGRES_DB=winedining
POSTGRES_USER=winedining
POSTGRES_PASSWORD=gabulja!
POSTGRES_SERVER=j12b202.p.ssafy.io
POSTGRES_PORT=5432
```

3. 설정 파일

NGINX : default.conf

- 설정 파일 위치

```
/home/ubuntu/nginx/conf.d/
```

- default.conf

```
server {
        root /usr/share/nginx/html;
        index index.html index.htm index.nginx-debian.html;
        server_name <SUBDOMAIN_DOMAIN>;  # 예: j12b202.p.ssafy.io
        client_max_body_size 50M;

    location / {
            try_files $uri $uri/ /index.html;
    }

    location /api {
            proxy_pass http://localhost:8080/api;
            proxy_set_header X-Real_IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_set_header Host $host;
            proxy_redirect off;
            charset utf-8;
    }

    location /oauth2 {
            proxy_pass http://localhost:8080;
            proxy_set_header X-Real-IP $remote_addr;
```

```
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Host $host;
        proxy_redirect off;
        charset utf-8;
    }

    location /login {
        proxy_pass http://localhost:8080;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Host $host;
        proxy_redirect off;
        charset utf-8;
    }

    location /.well-known/acme-challenge/ {
        allow all;
        root /var/www/certbot;
    }

    listen [::]:443 ssl ipv6only=on;
    listen 443 ssl;
    ssl_certificate /etc/letsencrypt/live/<SUBDOMAIN_DOMAIN>/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/<SUBDOMAIN_DOMAIN>/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;
}

server {
    root /usr/share/nginx/html;
    index index.html index.htm index.nginx-debian.html;
    server_name <MAIN_DOMAIN> www.<MAIN_DOMAIN>;
    client_max_body_size 50M;

    location / {
        try_files $uri $uri/ /index.html;
    }

    location /api {
        proxy_pass http://localhost:8080/api;
        proxy_set_header X-Real_IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Host $host;
        proxy_redirect off;
        charset utf-8;
    }

    location /oauth2 {
        proxy_pass http://localhost:8080;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Host $host;
        proxy_redirect off;
        charset utf-8;
```

```
    }

    location /login {
        proxy_pass http://localhost:8080;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header Host $host;
        proxy_redirect off;
        charset utf-8;
    }

    location /.well-known/acme-challenge/ {
        allow all;
        root /var/www/certbot;
    }

    listen [::]:443 ssl;
        listen 443 ssl;
        ssl_certificate /etc/letsencrypt/live/<MAIN_DOMAIN>/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/<MAIN_DOMAIN>/privkey.pem;
        include /etc/letsencrypt/options-ssl-nginx.conf;
        ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;
}

server {
    if ($host = <SUBDOMAIN_DOMAIN>) {
        return 301 https://$host$request_uri;
    }

    listen 80;
    listen [::]:80;
    server_name <SUBDOMAIN_DOMAIN>;
    return 404;
}

server {
    if ($host = www.<MAIN_DOMAIN>) {
        return 301 https://$host$request_uri;
    }

    if ($host = <MAIN_DOMAIN>) {
        return 301 https://$host$request_uri;
    }

    listen 80;
    listen [::]:80;
    server_name <MAIN_DOMAIN> www.<MAIN_DOMAIN>;
    return 404;
}
```

Jenkins : Jenkinsfile

- 설정 파일 위치 : 프로젝트 최상단
- Jenkinsfile

```
pipeline {
    agent any

    environment {
        BRANCH_NAME = "${GIT_BRANCH}"
        NODE_VERSION = 'node20'
        DEPLOY_PATH = '/home/ubuntu/nginx/html'
        JAVA_VERSION = 'jdk17'
        APP_NAME = 'react-container'
        DOCKER_IMAGE = 'react-container:latest'
        // FastAPI 관련 환경 변수
        // PYTHON_VERSION = 'python3.10'  // 사용할 Python 버전
        FASTAPI_APP_NAME = 'recommendation-api'
        FASTAPI_DOCKER_IMAGE = 'recommendation-api:latest'
        FASTAPI_PORT = '8000'  // FastAPI가 사용할 포트
    }

    stages {
        stage('Checkout') {
            steps {
                checkout scm
                script {
                    echo "현재 브랜치: ${BRANCH_NAME}"
                }
            }
        }

        stage('Backend Build & Deploy') {
            when {
                expression { BRANCH_NAME == 'origin/backend' }
            }
            tools {
                jdk "${JAVA_VERSION}"
            }
            steps {
                dir('backend') {
                    script {
                        sh '''
                            echo "===== Build Environment ====="
                            echo "JDK Version:"
                            java --version
                            echo "Docker Version:"
                            docker --version
                            echo "Current Directory:"
                            pwd
                            ls -la
                        '''

                        // Prepare Environment
                        sh '''
                            rm -rf src/main/resources
                            mkdir -p src/main/resources
                            chmod 777 src/main/resources
                        '''

                        // 시크릿 파일 설정 부분 (필요시 주석 해제)
                        withCredentials([
```

```
                file(credentialsId: 'prod-yaml', variable: 'prodFile'),
                file(credentialsId: 'secret-yaml', variable: 'secretFile')
            ]) {
            sh '''
                cp "$prodFile" src/main/resources/application-prod.yml
                cp "$secretFile" src/main/resources/application-secret.yml
                chmod 644 src/main/resources/application-*.yml
            '''
            }

            // Gradle 빌드
            sh '''
                chmod +x gradlew
                ./gradlew clean build -x test --no-daemon
            '''

            // Docker 배포
            sh '''
                docker rm -f ${APP_NAME} || true
                docker rmi ${DOCKER_IMAGE} || true
                docker build -t ${DOCKER_IMAGE} .
                docker run -d \
                  --name ${APP_NAME} \
                  -e SPRING_SERVLET_MULTIPART_MAX_FILE_SIZE=50MB \
                  -e SPRING_SERVLET_MULTIPART_MAX_REQUEST_SIZE=100MB \
                  --network my-network \
                  --restart unless-stopped \
                  -p 8080:8080 \
                  ${DOCKER_IMAGE}
              '''
          }
        }
    }
    post {
        success {
            echo '백엔드 빌드 및 배포 성공'
        }
        failure {
            echo '백엔드 빌드 및 배포 실패'
        }
    }

    stage('Frontend Build & Deploy') {
        when {
            expression { BRANCH_NAME == 'origin/frontend' }
        }
        tools {
            nodejs "${NODE_VERSION}"
        }
        steps {
            dir('frontend/winedining') {
                script {
                    // 빌드 전 상태 출력
                    sh '''
                        echo "===== Build Environment ====="
                        echo "Node Version:"
                        node --version
```

```
                echo "NPM Version:"
                npm --version
                echo "Current Directory:"
                pwd
                ls -la
            '''
            // 시크릿 파일 설정 부분 (필요시 주석 해제)
            withCredentials([
                file(credentialsId: 'react-env', variable: 'envFile')
            ]) {
            sh '''
                cp "$envFile" .env
                chmod 644 .env
            '''
            }

            sh '''
                echo "===== Starting Build Process ====="
                rm -rf node_modules
                npm install
                CI=false npm run build
            '''

            // 배포
            sh '''
                echo "===== Starting Deployment ====="
                echo "Cleaning deployment directory..."
                rm -rf ${DEPLOY_PATH}/*

                echo "Copying build files..."
                cp -r build/* ${DEPLOY_PATH}/

                echo "Verifying deployment..."
                ls -la ${DEPLOY_PATH}
            '''
        }
    }
}
post {
    success {
        echo '프론트엔드 빌드 및 배포 성공'
    }
    failure {
        echo '프론트엔드 빌드 및 배포 실패'
    }
}

stage('Recommendation API Build & Deploy') {
    when {
        expression { BRANCH_NAME == 'origin/recommendation' }
    }
    steps {
        dir('fastapi') {  // FastAPI 코드가 있는 디렉토리로 변경하세요
            script {
                sh '''
                    echo "===== Build Environment ====="
                    echo "Python Version:"
```

```
                python3 --version
                echo "Pip Version:"
                pip --version
                echo "Docker Version:"
                docker --version
                echo "Current Directory:"
                pwd
                ls -la
            '''

            // 가상환경 설정 및 의존성 설치
            sh '''
                python3 -m venv venv
                . venv/bin/activate
                pip install --upgrade pip
                pip install -r requirements.txt
            '''

            // 환경 변수 파일 설정 (필요한 경우)
            withCredentials([
                file(credentialsId: 'fastapi-env', variable: 'envFile')
            ]) {
                sh '''
                    cp "$envFile" .env
                    chmod 644 .env
                '''
            }

            // Docker 이미지 빌드 및 배포
            sh '''
                docker rm -f ${FASTAPI_APP_NAME} || true
                docker rmi ${FASTAPI_DOCKER_IMAGE} || true
                docker build -t ${FASTAPI_DOCKER_IMAGE} .
                docker run -d \\
                    --name ${FASTAPI_APP_NAME} \\
                    --network my-network \\
                    --restart unless-stopped \\
                    -p ${FASTAPI_PORT}:${FASTAPI_PORT} \\
                    ${FASTAPI_DOCKER_IMAGE}
            '''
                }
            }
        }
        post {
            success {
                echo '추천 API 빌드 및 배포 성공'
            }
            failure {
                echo '추천 API 빌드 및 배포 실패'
            }
        }
    }
}

post {
    success {
        script {
            def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
```

```
            def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
            withCredentials([string(credentialsId: 'mattermost-webhook', variable: 'WEBHOOK_URL')]) {
              mattermostSend(color: 'good',
                message: "빌드 성공: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<${
                endpoint: WEBHOOK_URL,
                channel: 'b202_'
              )
            }
          }
        }
        failure {
          script {
            def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
            def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
            withCredentials([string(credentialsId: 'mattermost-webhook', variable: 'WEBHOOK_URL')]) {
              mattermostSend(color: 'danger',
                message: "빌드 실패: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}(${Author_Name})\n(<${
                endpoint: WEBHOOK_URL,
                channel: 'b202_'
              )
            }
          }
        }
      }
    }
```

Dockerfile : Backend

- 설정 파일 위치

```
backend/Dockerfile
```

- Dockerfile

```
FROM openjdk:17-jdk

ARG JAR_FILE=build/libs/*.jar
# jar 파일 복제
COPY ${JAR_FILE} app.jar


# 실행 명령어 test
ENTRYPOINT ["java", "-Dspring.profiles.active=prod,secret", "-jar", "app.jar"]
```

Dockerfile : FastAPI

- 설정 파일 위치

```
fastapi/Dockerfile
```

- Dockerfile

```


# 1. Python 이미지를 베이스로 사용
FROM python:3.12
# 2. 작업 디렉토리 설정
```

```
WORKDIR /app
# 3. 의존성 파일을 컨테이너에 복사
COPY requirements.txt /app/
# 4. 의존성 설치
RUN pip install --no-cache-dir -r requirements.txt
# 5. .env 파일 복사 (추가된 부분)
COPY .env /app/
# 6. FastAPI 애플리케이션 코드 복사
COPY app /app/app
# 7. FastAPI 애플리케이션 실행 명령어
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

2. 배포 방법

1. EC2 접속

1. window teminal 설치

   www.microsoft.com/ko-kr/p/windows-terminal/9n0dx20hk701?activetab=pivot:overviewtab

2. window openSSH 사용 설정 (재부팅)
3. aws 접속하여 인스턴스 접속정보 확인

   ssh -i I12B208T.pem ubuntu@i12b208.p.ssafy.io

4. 프라이빗 키(pem 키) 설정
5. window terminal에서 ssh 로 ec2 접속하기

2. 방화벽 설정

포트허용

로컬에서 개발중인 프론트엔드의 접근을 위해 8080을 열어주었습니다.

9090은 젠킨스 포트로 쓰기위해 열었습니다.

https 접근을 위해 443포트도 열었습니다.

```
sudo ufw allow 8080
sudo ufw allow 9090
sudo ufw allow 443
sudo ufw status numbered
```

3. 도커 설치

java 설치 (springboot프로젝트 버전)

```
# Java 17 설치
sudo apt update
sudo apt install openjdk-17-jdk -y

# 설치된 버전 확인
java -version
```

도커설치

```
$ sudo apt-get update

# 패키지 인덱스 업데이트
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common

# Docker의 공식 GPG 키 추가
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

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

# 패키지 인덱스 업데이트
$ sudo apt-get update

# Docker CE 설치
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin


# Docker 서비스 시작 및 자동 시작 설정
$ sudo systemctl start docker
$ sudo systemctl enable docker

# Docker 그룹에 사용자 추가
$ sudo usermod -aG docker ${USER}

# 확인
$ docker --version
```

스왑메모리 지정

현재 사용중인 메모리 양(16GB)의 두배(32GB)로 설정

→ 하나의 EC2에서 여러 어플리케이션 서버를 동작하는 경우 필수

```
# 시스템 업데이트
sudo apt update

# Swap Space 생성하기
sudo fallocate -l 32G /swapfile
sudo chmod 600 /swapfile
sudo mkswap /swapfile

# RAM swap 하기
sudo swapon /swapfile

# 확인
sudo swapon --show
free -h
               total        used        free      shared  buff/cache   available
```

```
Mem:           15Gi       12Gi      300Mi      1.0Mi      3.3Gi      3.2Gi
Swap:          31Gi        0B        31Gi
```

4. Jenkins 설치

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

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

# 패키지 목록 업데이트
sudo apt-get update

# Jenkins 설치
sudo apt-get install jenkins

# 확인
$ sudo systemctl status jenkins

# jenkins 권한 주기
sudo usermod -aG docker jenkins

$ sudo systemctl start jenkins

# 초기 비밀번호 확인
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

Jenkins 환경설정 변경

```
# 젠킨스 설정 변경을 위해 서비스 중지
sudo systemctl stop jenkins

# jenkins 설치 경로로 이동
cd /var/lib/jenkins

# update-center에 필요한 CA파일을 다운로드하고 권한을 변경합니다.
sudo mkdir update-center-rootCAs
sudo wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-center/rootCA/update-center.crt -O \
./update-center-rootCAs/update-center.crt
sudo chown -R jenkins:jenkins update-center-rootCAs

# default 설정에서 특정 미러사이트로 대체하도록 아래 명령어를 실행
sudo sed -i 's#https://updates.jenkins.io/update-center.json#https://raw.githubusercontent.com
/lework/jenkins-update-center/master/updates/tencent/update-center.json#' ./hudson.model.UpdateCenter.xml

# URL이 위에꺼로 바뀌었는지 확인
cat hudson.model.UpdateCenter.xml

# !다시 jenkins 구동하기
sudo systemctl restart jenkins
```

Jenkins 포트 변경

/usr/lib/systemd/system/jenkins.service에 위치한 Environment-"JENKINS_PORT=8080" 부분을 9090으로 변경하면된다.

```
sudo vi /usr/lib/systemd/system/jenkins.service
```

재실행 하기

```
sudo systemctl daemon-reload

# 재시작
sudo systemctl restart jenkins
```

5. Jenkins 접속

이제 `http://ip주소:9090`로 들어가서 초기비밀번호 입력 후

jenkins id, pw, name, email 작성

`suggested plugins`를 설치한 후 admin 생성하고 URL 또한 `http://ip주소:9090`으로 설정

> [설치 플러그인]

- Generic Webhook Trigger
- Gitlab
- Gitlab API
- Gitlab Authentication
- Mattermost Notification
- Docker pipeline

6. Gitlab과 Jenkins 연동

Gitlab에서 토큰 발급

Gitlab 프로젝트에 접속해서 Settings관리 → Access Tokens

Select a role → Maintainer

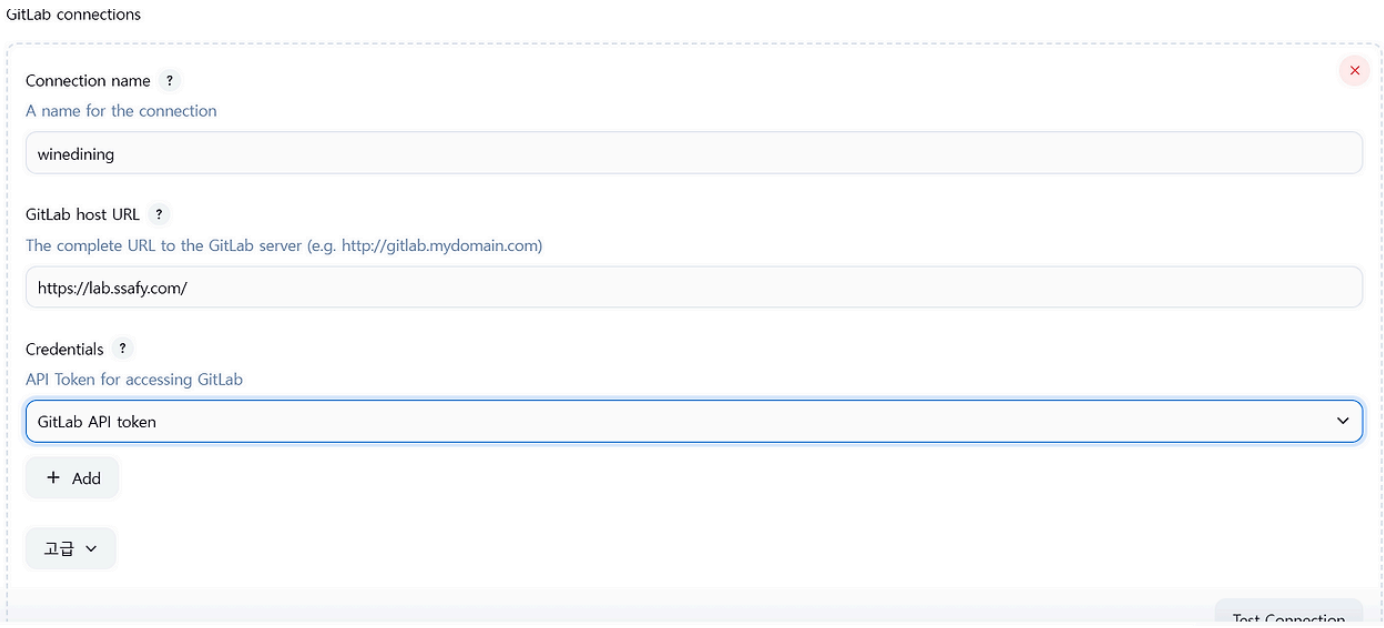
Select scopes → api, read_api, read_repository 체크

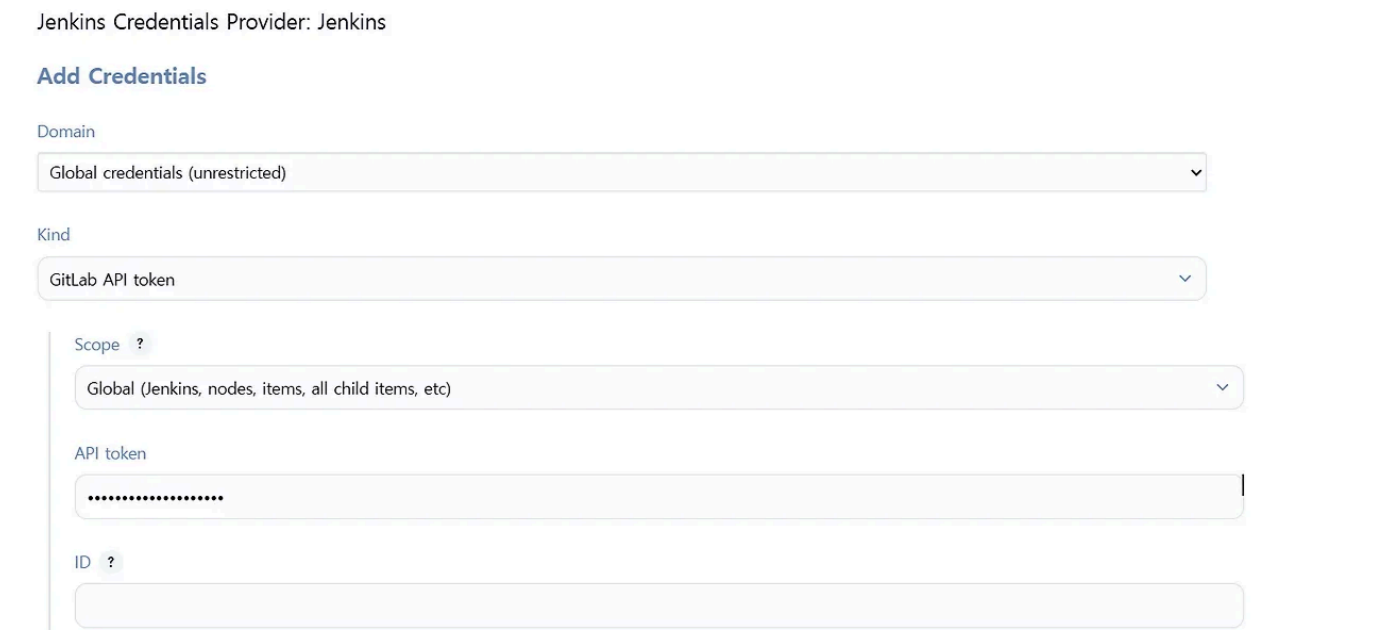
Jenkins에서 Gitlab 연결

Jenkins 관리 → System 설정 → Gitlab탭에서 다음과 같이 설정

- Connection name : 원하는 이름 입력
- Gitlab host URL : 깃랩 메인 주소 입력
- Credentials : +Add 버튼을 눌러 Credential 추가페이지로 이동
- Kind : GitLab API token
- Scope : Global
- API token : 깃랩에서 생성한 엑세스 토큰 입력

id, description은 설정해주지 않아도 괜찮다.

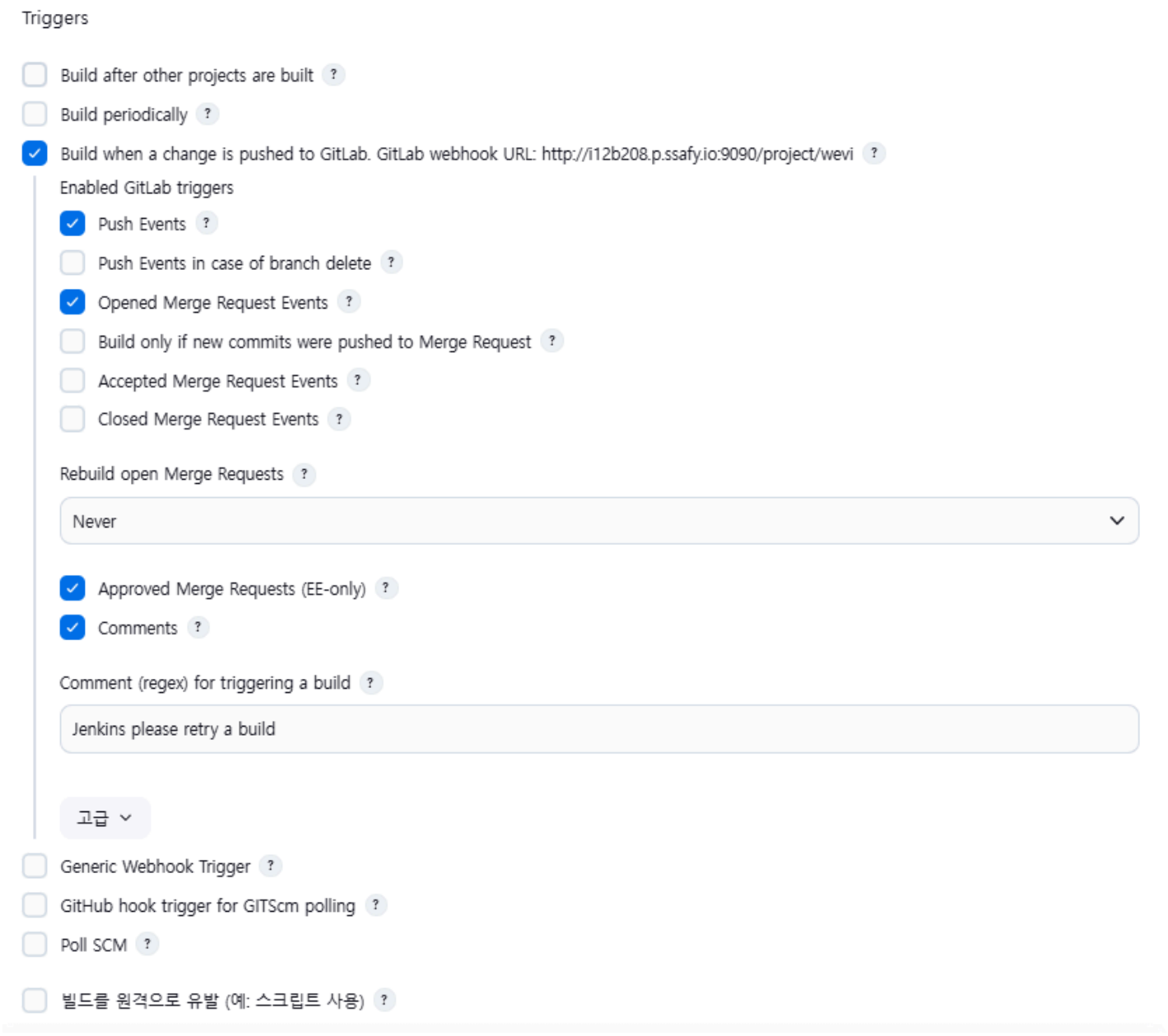




Webhook 설정

jenkins

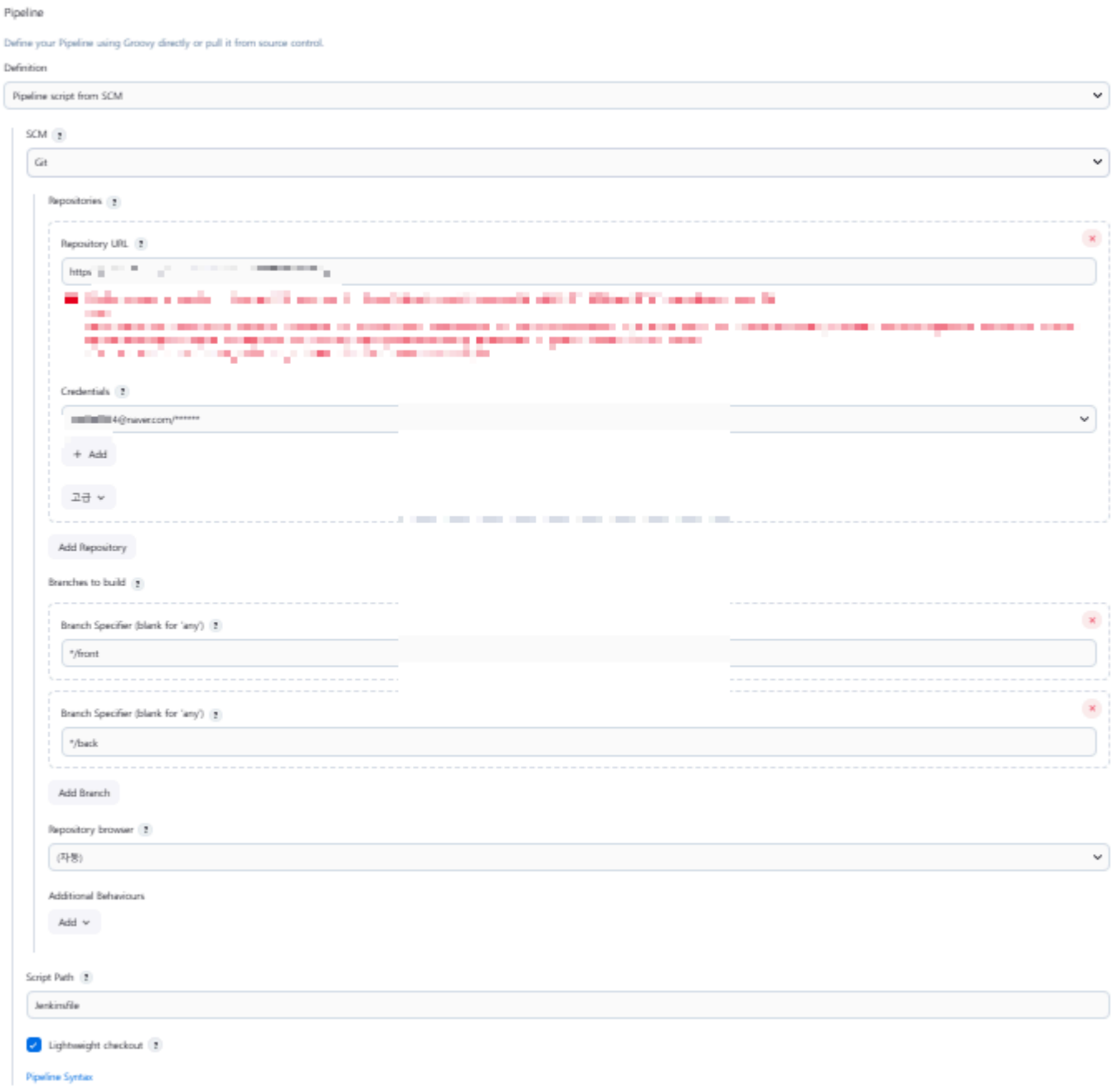
이제 깃랩에서 특정 브랜치에 업데이트가 되면 자동으로 젠킨스가 감지하여 빌드할 수 있도록 Web-hook을 설정



1. Jenkins에서 '새로운 Item' → 'Pipeline'
2. 'Build Triggers'에서 'Build when a change is push to Gitlab~'
3. 뒤에 URL 기록해두고
4. 고급 버튼 에서 `Secret token`에서 Generate해서 토큰 저장

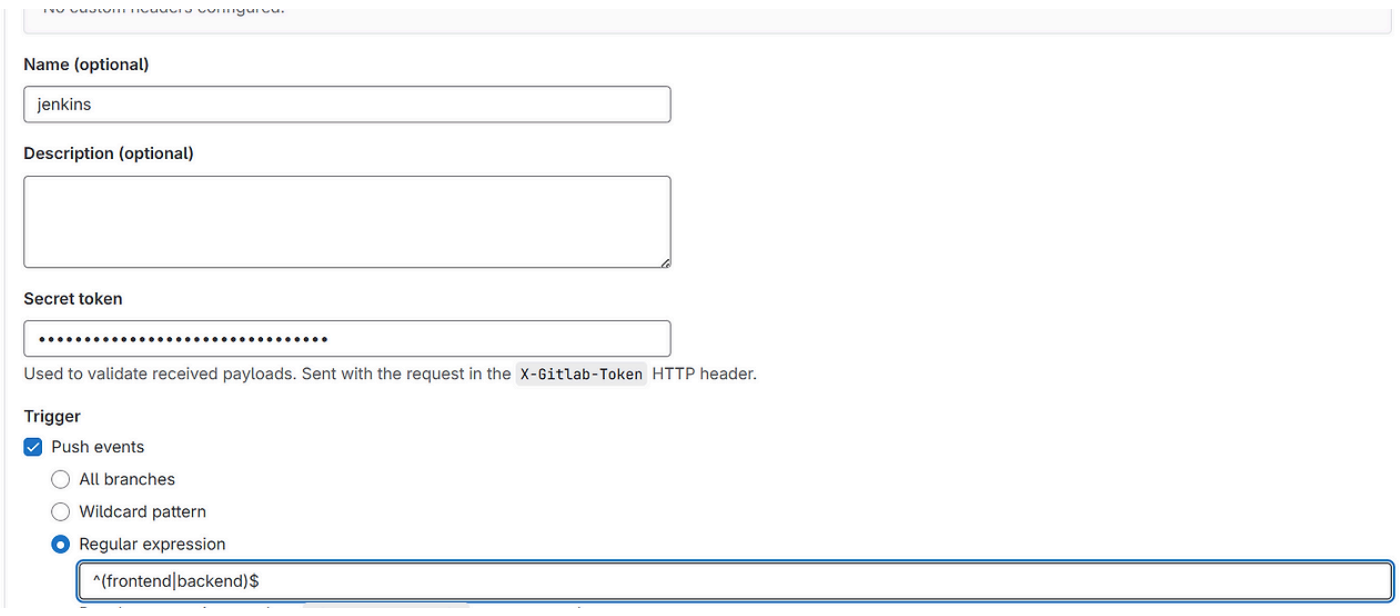
- Branch Specifier를 `*/master`에서 `*/front`, `*/back`으로 변경

SCM에서 Git 주소를 입력하고 Credentials에 아이디와 비밀번호를 적용해야 함 (토큰 사용 불가)



Gitlab 설정

1. webhook탭에서 기록한 url과 secret toekn 넣음
2. branch는 정규식 브랜치 선택해서 `^(front|back)$` 넣음



7. 프론트엔드(React) 연결

React를 빌드한 결과물을 Nginx의 정적파일로 마운트하여 사용. Nginx에 https 적용, `/api` uri는 localhost:8080포트로 프록시하도록 적용

- **순서**

nginx 설치 -> nginx 설정 수정 -> Https 발급(Let's Encrypt) -> nginx 중지 -> Docker로 nginx 생성 후 실행(기존 nginx설정, https 파일 마운트)

Nginx 설치

```
# Nginx 설치
sudo apt update
sudo apt install nginx

# Nginx 상태 확인
sudo systemctl status nginx
```

Nginx 설정 수정

`sudo vi /etc/nginx/sites-available/default`

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    server_name 도메인이름(ex.tripggukgguk.site);

    root /var/www/html;
    index index.html;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

Nginx 시작

```
# nginx 설정 문법 체크
sudo nginx -t

sudo systemctl start nginx
sudo systemctl enable nginx
```

https 적용

자동 갱신은 적용하지 않았음

```
# certbot 설치
sudo apt install certbot python3-certbot-nginx

# 인증서 발급 (자동으로 Nginx 설정도 수정됨)
sudo certbot --nginx -d 도메인이름
```

Docker로 전환

```
# 디렉토리 생성
mkdir -p /home/ubuntu/nginx/html
mkdir -p /home/ubuntu/nginx/conf

# 소유권 설정
sudo chown -R ubuntu:ubuntu /home/ubuntu/nginx
```

```
# 호스트의 nginx 중지
sudo systemctl stop nginx
sudo systemctl disable nginx
```

Docker Nginx 설정

```
sudo mkdir -p /home/ubuntu/nginx/conf.d
sudo chown -R ubuntu:ubuntu /home/ubuntu/nginx
sudo cp /etc/nginx/sites-available/default /home/ubuntu/nginx/conf.d/default.conf
sudo vi /home/ubuntu/nginx/conf.d/default.conf
```

root 경로를 `/usr/share/nginx/html`로 변경 (도커 컨테이너 내부 경로)

Nginx 컨테이너 실행

```
# 이미지 받기
docker pull nginx

docker run -d --name nginx \
--network host \
-v /home/ubuntu/nginx/conf.d:/etc/nginx/conf.d:ro \
-v /etc/letsencrypt:/etc/letsencrypt:ro \
-v /home/ubuntu/nginx/html:/usr/share/nginx/html \
nginx:stable

# 테스트
echo "<h1>HTTPS Test</h1>" > /home/ubuntu/nginx/html/index.html
```

- v는 마운팅nginx 설정 파일과, https파일
- -network 해야지, localhost호출이 가능함(안하면 docker와 호스트의 network가 따로)

트러블 슈팅

프론트엔드만 실행하기로 결정 후 Node.js를 찾을 수 없음

1. EC2에서 Node.js 직접 설치

   ```
   curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
   sudo apt install -y nodejs
   ```

2. Node.js가 설치되어 있지만 `sudo su - jenkins` 상태에서 `node --version`이 작동하지 않는 경우 Jenkins 실행 계정의 `PATH` 설정 필요

   a. Node.js 설치 경로 확인

   ```
   which node
   ```

   b. Jenkins 사용자 환경변수 설정

   ```
   sudo nano /etc/default/jenkins
   ```

   맨 아래에 다음 줄 추가:

   ```
   PATH=$PATH:/usr/bin
   export PATH
   ```

c. Jenkins 재시작

```
sudo systemctl restart jenkins
```

Node.js 18버전 호환 불가, 20버전으로 업그레이드

- React Router가 Node.js 20 이상을 요구함
- 현재 설치된 Node.js 버전이 `v18.20.6`이라 호환되지 않음

Node.js 20으로 업데이트

```
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
sudo apt install -y nodejs
node --version
```

플러그인을 설치했으면, node20을 Jenkins에 등록

배포 경로 문제 해결

- Jenkins는 `jenkins` 사용자의 권한으로 실행되므로 `/home/ubuntu/nginx/html/` 접근 시 권한 문제 발생 가능
- 파일 권한을 조정하여 `jenkins` 사용자도 접근 가능하도록 변경
- jenkins 파일에 적용완료

```
sudo chown -R jenkins:jenkins /home/ubuntu/nginx/html/
sudo chmod -R 755 /home/ubuntu/nginx/html/
```

새로고침 시 404 오류 발생

- 웹 서버 리다이렉션 설정 필요
- Nginx 설정 예시:

```
location / {
    try_files $uri $uri/ /index.html;
}
```

8. 백엔드(Springboot) 연결

nginx 설정 변경 /api

docker에 띄워져있는 nginx에 주소/api 로 온다면 localhost:8080으로 보내주기 위해 환경변수 파일 변경

`sudo vi /home/ubuntu/nginx/conf.d/default.conf`

```
server {
    listen 80;
    server_name _;  # 여기에 실제 도메인 입력

    root /usr/share/nginx/html;
    index index.html;

    location / {
        try_files $uri $uri/ /index.html;
    }

    # API 요청을 백엔드로 프록시
    location /api {
        proxy_pass http://localhost:8080;  # 스프링 부트 서버 주소
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
```

```
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
    }
}
```

```
# 도커 재시작
docker restart nginx
```

gitlab credential 설정

숨기려는 파일 업로드

`Jenkins 관리` -> `Credentials` → `Stores scoped to Jenkins의 (glabal)` -> `Add Credentails`

`Secret file` 선택 후 업로드

Jenkins file에 적용된 부분:

```
stage('Secrets Setup') {
    steps {
        withCredentials([
            file(credentialsId: 'prod-yaml', variable: 'prodFile'),
        ]) {
            sh '''
                cp "$prodFile" src/main/resources/application-prod.yml
                chmod 644 src/main/resources/application-*.yml
            '''
        }
    }
}
```

Docker file

```
FROM openjdk:17-jdk-slim

ARG JAR_FILE=build/libs/*.jar
# jar 파일 복제
COPY ${JAR_FILE} app.jar

# 실행 명령어
ENTRYPOINT ["java", "-Dspring.profiles.active=prod,secret", "-jar", "app.jar"]
```

JDK 설정

9. Mattermost와 연결

Credentials 설정

Jenkins 파일에서 숨길 mattermost url 과 이메일값을 숨기기 위해 Credential Secret text를 사용하였습니다.

방법

Jenkins 관리 → Credentials → Credentail 추가 → Secret text → Secret에 숨길 키값 적기 → ID에 호출할 ID값 적기

Jenkinsfile에 적용 부분

```
post {
    success {
        script {
            def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
            def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
            mattermostSend(color: 'good',
                message: "빌드 성공: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}
                (${Author_Name})\n(<${env.BUILD_URL}|Details>)",
                endpoint: '',
                channel: ''
                    )
        }
    }
    failure {
        script {
            def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).trim()
            def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true).trim()
            mattermostSend(color: 'danger',
                message: "빌드 실패: ${env.JOB_NAME} #${env.BUILD_NUMBER} by ${Author_ID}
                (${Author_Name})\n(<${env.BUILD_URL}|Details>)",
                endpoint: '',
                channel: ''
                    )
        }
    }
}
```

10. DB 연결

도커를 이용해서 PostgreSQL과 Redis를 띄우고 docker network를 이용하여 같은 네트워크로 통신 할 수 있도록 구축하였습니다.

Docker network

`--link`는 Docker가 공식적으로 비추천하는 기능이므로, 대신 Docker 네트워크를 사용하는 것이 좋습니다. `docker network create` 명령어로 새로운 네트워크를 생성하고 두 컨테이너가 같은 네트워크 내에서 실행되도록 설정합니다.

```
# docker network 생성
docker network create my-network

# MySQL 컨테이너 실행 시 네트워크 지정
docker run -d --name mysql-container --network my-network -p 3306:3306 -v mysql-volume:/var/lib/mysql -e MYSQL_R

# Spring Boot 애플리케이션 컨테이너 실행 시 동일한 네트워크 지정 -> Jenkins file에 설정완료
```

```
docker run -d --name ${APP_NAME} --network my-network --restart unless-stopped -p 8080:8080 ${DOCKER_IMAGE}
```

위와 같이 `--network my-network` 옵션을 사용해 두 컨테이너가 같은 네트워크에 속하게 하면, MySQL 컨테이너의 이름인 `mysql-container`로 접근할 수 있습니다.

PostgreSQL 설치

```
# Docker images
docker pull postgres:16.8

# Docker Volume
docker volume create postgres-volume

# Docker 실행
docker run -d --name postgres-container --network my-network -p 5432:5432 \
  -v postgres-volume:/var/lib/postgresql/data \
  -e POSTGRES_USER=postgres \
  -e POSTGRES_PASSWORD=<비밀번호> \
  postgres:16.8

# postgres 유저로 기본 DB 접속
docker exec -it postgres-container psql -U postgres

# 특정 유저로 특정 DB 접속
docker exec -it postgres-container psql -U <유저이름> -d <DB이름>

# 데이터베이스 생성
CREATE DATABASE <DB이름>;

# 유저 생성
CREATE USER <유저이름> WITH PASSWORD '<비밀번호>';

# 기본 설정 변경
ALTER ROLE <유저이름> SET client_encoding TO 'UTF8';
ALTER ROLE <유저이름> SET default_transaction_isolation TO 'read committed';
ALTER ROLE <유저이름> SET timezone TO 'Asia/Seoul';

# 권한 부여
GRANT ALL PRIVILEGES ON DATABASE <DB이름> TO <유저이름>;
GRANT ALL ON SCHEMA public TO <유저이름>;
ALTER DEFAULT PRIVILEGES IN SCHEMA public GRANT ALL ON TABLES TO <유저이름>;

# DB 오너 변경
ALTER DATABASE <DB이름> OWNER TO <유저이름>;

# nano 설치
apt update && apt install -y nano

# 인증 설정 파일 수정
nano /var/lib/postgresql/data/pg_hba.conf

# 변경 전
host all all all trust

# 변경 후
host all all all scram-sha-256
```

```
# 도커 재시작
docker restart postgres-container
```

Redis 설치

```
# Docker images
docker pull redis

# Docker Volume
docker volume create redis-volume

# Docker 실행
docker run -d \
  --name redis-container \
  --network my-network \
  -p 6379:6379 \
  --restart unless-stopped \
  redis:latest redis-server --requirepass <비밀번호> \
  --appendonly yes


# Redis CLI로 접속 테스트
docker exec -it redis-container redis-cli

# 비밀번호 인증
auth <비밀번호>

# 간단한 SET/GET 테스트
set test_key "Hello Redis"
get test_key
```

11. FastAPI 연결

FastAPI 또한 CI/CD 가 가능하도록 구축하였습니다.

Jenkins 와 Gitlab에서 recommend 브랜치 Webhook 설정

jenkins

깃랩에서 recommend 브랜치에 업데이트가 되면 자동으로 젠킨스가 감지하여 빌드할 수 있도록 Web-hook을 설정

1. Jenkins에서 '새로운 Item' → 'Pipeline'
2. 'Build Triggers'에서 'Build when a change is push to Gitlab~'
3. 뒤에 URL 기록해두고
4. 고급 버튼 에서 `Secret token`에서 Generate해서 토큰 저장

- Branch Specifier를 `*/master`에서 `*/front`, `*/back`, `*/recommend`으로 변경

Gitlab 설정

1. webhook탭에서 기록한 url과 secret toekn 넣음
2. branch는 정규식 브랜치 선택해서 `^(front|back|recommend)$` 넣음

Jenkins에 Python 도구 설정 추가

1. **Python 플러그인 설치**

- `Jenkins 관리` → `플러그인 관리` → `사용 가능` 탭
- **ShiningPanda Plugin** 검색 → 체크 후 설치
- Jenkins 재시작

2. **Python 도구 설정**

- `Jenkins 관리` → `글로벌 도구 설정`
- "Python" 섹션에서 **Python 설치 추가**
  - 이름: `Python3.10` 등
  - 설치 방식:
    - 시스템 Python 사용 시: `시스템 Python` 선택 + 경로 지정
    - 또는 자동 설치 선택
- 설정 후 **저장**

Jenkinsfile 설정

```
stage('Recommendation API Build & Deploy') {
    when {
        expression { BRANCH_NAME == 'origin/recommendation' }
    }
    steps {
        dir('recommendation') {  // FastAPI 코드가 있는 디렉토리
            script {
                // 환경 변수 파일 설정
                withCredentials([
                    file(credentialsId: 'fastapi-env', variable: 'envFile')
                ]) {
                    sh '''
                        # .env 파일 복사
                        cp "$envFile" .env
                        chmod 644 .env

                        # Docker 빌드 및 배포
                        docker rm -f ${FASTAPI_APP_NAME} || true
                        docker rmi ${FASTAPI_DOCKER_IMAGE} || true
                        docker build -t ${FASTAPI_DOCKER_IMAGE} .
                        docker run -d \\
                        --name ${FASTAPI_APP_NAME} \\
                        --network my-network \\
                        --restart unless-stopped \\
                        -p ${FASTAPI_PORT}:${FASTAPI_PORT} \\
                        ${FASTAPI_DOCKER_IMAGE}
                    '''
                }
            }
        }
    }
    post {
        success {
            echo '추천 API 빌드 및 배포 성공'
        }
        failure {
            echo '추천 API 빌드 및 배포 실패'
        }
```

```
  }
}
```

Dockerfile 작성

```
# 1. Python 이미지를 베이스로 사용
FROM python:3.12

# 2. 작업 디렉토리 설정
WORKDIR /app

# 3. 의존성 파일을 컨테이너에 복사
COPY requirements.txt /app/

# 4. 의존성 설치
RUN pip install --no-cache-dir -r requirements.txt

# 5. .env 파일 복사
COPY .env /app/

# 6. FastAPI 애플리케이션 코드 복사
COPY app /app/app

# 7. FastAPI 애플리케이션 실행 명령어
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```