

포팅 매뉴얼

목차

1. 프로젝트 개발 환경
2. 환경 변수
3. Front-End Build
4. Back-End Build
5. EC2 설정
6. Jenkins 설정
7. Jenkins Build 완료 화면
8. QingDao OnlineJudge 설치
9. Back-End DockerFile
10. Front-End DockerFile
11. Front-End Nginx Config 파일
12. Port 정리

1. 프로젝트 개발 환경

형상 관리

- GitLab

이슈 관리

- Jira

커뮤니케이션

- Mattermost
- Webex
- Notion

OS

- Windows 10

UI/UX

- Figma

IDE

- IntelliJ Build #IU-223.8214.52, built on December 20, 2022
- Visual Studio Code Version 10.0.19045.2364

DataBase

- MySQL 8.0.32
- Redis Server 7.0.8

Server

- AWS EC2
 - Ubuntu 20.04 LTS
 - Docker 20.10.12

배포

- Jenkins 2.375.1

Front-End

- React 18.2.0
- Node.js 18.13.0
- TypeScript 7.20
- Ant-design 5.1.5
- recoil 0.7

Back-End

- OpenJDK 1.8
- SpringBoot 2.7.7
- Spring Framework 5.3.24
- Spring Security 5.7.6
- Spring NetFlix Eureka 1.10.17
- Spring Cloud 2021.0.5

2. 환경 변수

apigateway-service

- EUREKA_CLIENT_SERVICE_URL_DEFAULT_ZONE # 넷플릭스 유레카 서버
- ACCESS_TOKEN_EXPIRATION_TIME # JWT 액세스 토큰 만료 시간
- ACCESS_TOKEN_SECRET # JWT 액세스 토큰 시크릿 키
- REFRESH_TOKEN_EXPIRATION_TIME # JWT 리프레시 토큰 만료 시간
- REFRESH_TOKEN_SECRET # JWT 리프레시 토큰 시크릿 키

user-service 의 application.yml 파일

- AWS_S3_BUCKET # AWS 버킷(사진 저장 공간) 이름
- AWS_CREDENTIAL_ACCESS_KEY # AWS 버킷 액세스 키
- AWS_CREDENTIAL_SECRET_KEY # AWS 버킷 시크릿 키
- AWS_REGION_STATIC # AWS 버킷 지역

rank-service 의 application.yml 파일

- SPRING_DATASOURCE_DRIVER_CLASS_NAME # 데이터베이스 드라이버 클래스
- SPRING_DATASOURCE_URL # 데이터베이스 데이터소스 URL
- SPRING_REDIS_HOST # 레디스 호스트
- SPRING_REDIS_PORT # 레디스 포트
- SPRING_REDIS_PASSWORD # 레디스 비밀번호

3. Front-End Build

로컬 환경에서의 실행

1. 프로젝트 위치에서 `npm i` 를 입력하여 package 설치
2. `npm run start` 명령어로 로컬 환경에서 실행

```
npm i
npm run start
```

배포 환경을 위한 빌드

1. 프로젝트 위치에서 `npm i` 를 입력하여 package 설치
2. `npm run build` 명령어로 build 결과물 추출

```
npm i
npm run build
```

4. Back-End Build

로컬 환경에서의 실행

1. apigateway-service 실행
2. match-service 실행
3. user-service 실행

4. log-service 실행
5. problem-service 실행
6. battle-service 실행
7. rank-service 실행
8. discovery-service 실행

배포 환경을 위한 빌드

1. 각 서비스의 디렉토리로 이동
2. `./gardlew clean build -x test` 명령어로 빌드

```
./gardlew clean build -x test
```

3. 빌드폴더로 이동, `java -jar /app.jar` 명령어로 build 된 jar 파일 실행

```
java -jar /app.jar
```

5. EC2 설정

1. apt 업데이트

```
apt-get update
```

2. Docker 설치

```
apt-get install docker.io
```

3. 도커 데몬 실행

```
systemctl start docker
```

4. Docker Jenkins Image 다운로드

```
docker pull jenkins/jenkins:2.375.1
```

5. `docker run -d --name jenkins --env JENKINS_OPTS = --httpPort=8888 -p 8888:8888 -v /jenkins:/var/jenkins_home -v /usr/bin/docker:/usr/bin/docker -v /var/run/docker.sock:/var/run/docker.sock -u root jenkins/jenkins:2.375.1`

위 명령어를 통해 젠킨스 컨테이너의 이름을 jenkins로 설정, 젠킨스의 내부 포트번호를 8888번으로 설정 후 외부 포트 8888번과 포트포워딩, 젠킨스 컨테이너 내부의 /var/jenkins_home 폴더와 EC2 의 /jenkins 폴더를 볼륨 마운트, Jenkins 내부에서 Docker를 사용하기 위해 (Docker out of Docker) 볼륨 마운트, jenkins/jenkins:2.375.1 도커 이미지로 도커 컨테이너 실행

6. Docker Redis Image 다운로드

```
docker image pull redis
```

7. Redis 데이터 백업을 위해 볼륨 마운트, 외부포트 6000번과 외부포트 6000번을 포트포워딩, /data/redis.conf 파일을 옵션으로 redis 실행

```
docker run -v /redis:/data --name spring-redis -d -p 6000:6000 redis redis-server /data/redis.conf
```

```
port 6000
bind 0.0.0.0
requirepass /*레디스 비밀번호 설정*/
```

8. Docker MySQL Image 다운로드

```
docker image pull mysql
```

9. MySQL 실행

```
docker run --name mysql -e MYSQL_ROOT_PASSWORD={루트 패스워드 입력} -d -p 3306:3306 mysql
```

6. Jenkins 설정

1. <http://ip:6000/> 로 젠킨스 홈페이지 진입 후 패스워드창에 docker log jenkins 를 입력해서 얻어온 키값 입력

```
*****
*****
*****

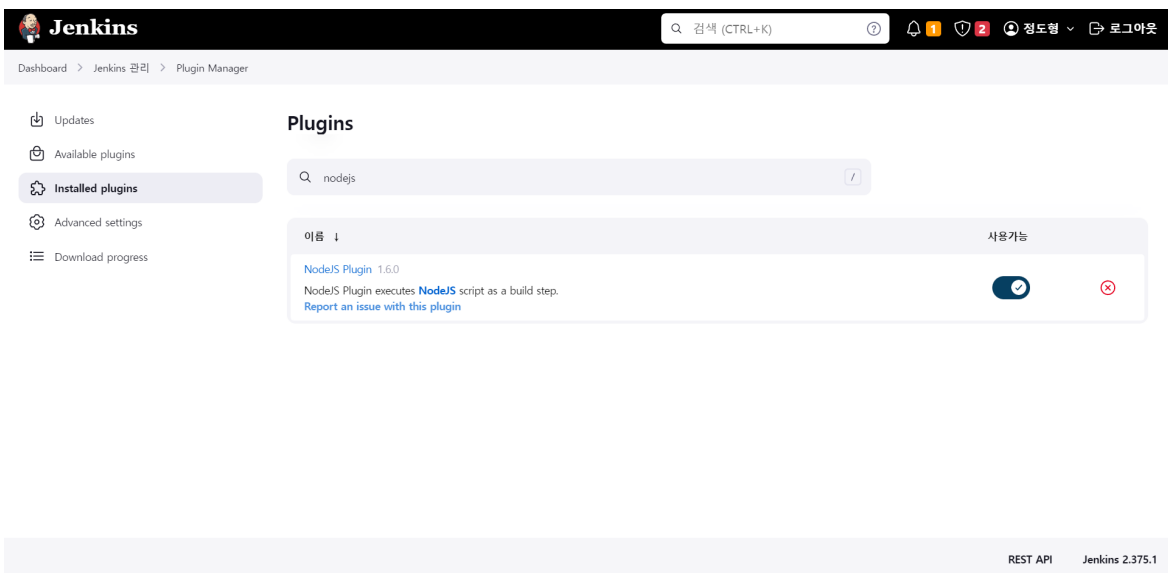
Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

이곳에 키 값이 나옵니다.

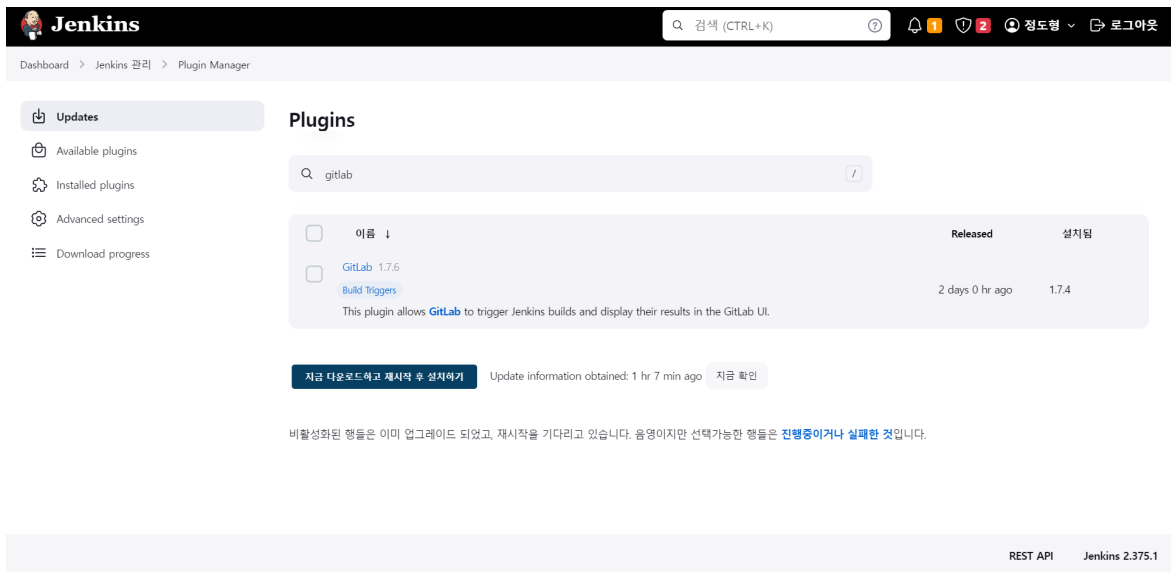
This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*****
*****
*****
```

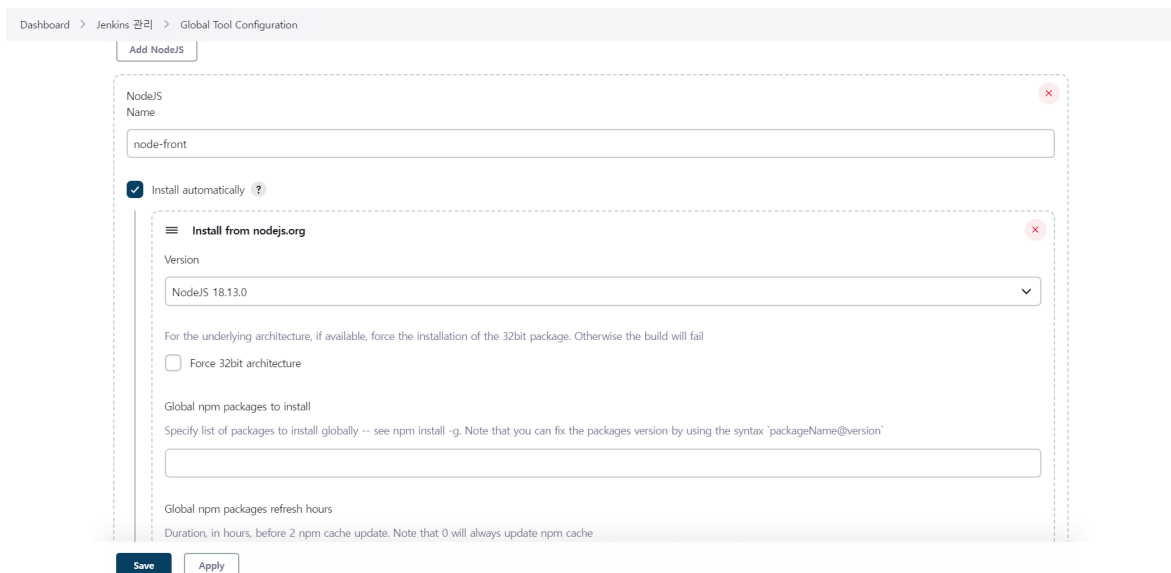
2. Jenkins 관리 → Plugin Manager에서 NodeJS Plugin 설치



3. Jenkins 관리 → Plugin Manager에서 GitLab 관련 Plugin 설치



4. Jenkins 관리 → Global Tool Configuration에서 NodeJS 항목 추가



5. WebHook을 위한 Credential 설정

- ID는 GitLab ID
- Password는 Gittlab의 Project access Token

Dashboard > Jenkins 관리 > Credentials > System > Global credentials (unrestricted) > wjdhgud5769/***** (jenkins_project)

Update
Delete
Move

Update credentials

Scope ?

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

Username ?

☐ Treat username as secret ?

Password ?

Concealed Change Password

ID ?

jenkins_project

Description ?

jenkins_project

Save

6. Jenkins 새로운 Item 만들기

- jenkins 프로젝트 이름 입력 후 Pipeline으로 생성
- Back-End 프로젝트 갯수만큼 Item 생성

Jenkins

Dashboard > All >

Enter an item name

> This field cannot be empty, please enter a valid name

Freestyle project

이것은 Jenkins의 주요 기능입니다. Jenkins은 어느 빌드 시스템과 어떤 SCM(항상관리)으로 묶인 당신의 프로젝트를 빌드할 것이고, 소프트웨어 빌드보다 다른 어떤 것에 자주 사용될 수 있습니다.

Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Multi-configuration project

다양한 환경에서의 테스트, 플러그인 특성 빌드, 기타 종종 커진 다수의 서로다른 환경설정이 필요한 프로젝트에 적합함.

Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

Multibranch Pipeline

Creates a set of Pipeline projects according to detected branches in one SCM repository.

Organization Folder

Creates a set of multibranch project subfolders by scanning for repositories.

If you want to create a new item from other existing, you can use this option:

7. Item GitLab 설정

- Gitlab webhook 옵션 체크

Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ Build when a change is pushed to GitLab. GitLab webhook URL:

- 하단의 [고급] 버튼 클릭 후 Generate 버튼을 클릭하여 Secret Key 생성 후 저장

Secret token ?

Generate

8. 파이프라인 스크립트 입력(Back-End)

```
pipeline {
  agent any

  stages {
    stage('Pull') {
      steps {
        git branch: '{WebHook을 설정한 브랜치를 입력하세요.}', credentialsId: '{생성한 Credential의 ID를 입력하세요.}', url: '{이곳에 (
      }
    }

    stage('springboot build'){
      steps {
        dir('{프로젝트의 빌드 파일이 있는 경로를 입력하세요.}'){
          sh 'ls -l'
          sh 'chmod +x gradlew'
          sh './gradlew clean build -x test'
        }
      }
    }

    stage('Build') {
      steps {
        dir('{프로젝트의 빌드 파일이 있는 경로를 입력하세요.}'){
          sh 'docker build -t {생성할 도커 이미지의 이름을 입력하세요.} ./'
        }
      }
    }

    stage('Deploy') {
      steps{
        sh 'docker ps -f name={컨테이너 이름} -q | xargs --no-run-if-empty docker container stop'
        sh 'docker container ls -a -f name={컨테이너 이름} -q | xargs -r docker container rm'
        sh 'docker images --no-trunc --all --quiet --filter="dangling=true" | xargs --no-run-if-empty docker rmi'
        sh 'docker run -d --name {실행할 컨테이너 이름} -p 8000:8000 {생성한 도커 이미지의 이름}'
      }
    }

    stage('Finish') {
      steps{
        sh 'docker images -qf dangling=true | xargs -I{} docker rmi {}'
      }
    }
  }
}
```

9. 위와 같은 과정으로 Item 생성 후 파이프라인 스크립트 입력(Back-End)

```
pipeline {
  agent any

  tools {
    nodejs "{Plugin 설치 후 설정한 Nodejs의 이름}"
  }

  stages {
    stage('Pull') {
      steps {
        git branch: '{WebHook을 설정한 브랜치를 입력하세요.}', credentialsId: '{생성한 Credential의 ID를 입력하세요.}', url: '{이곳에 (
      }
    }

    stage('React Build') {
      steps {

```

```

    dir('{리액트 빌드 폴더의 경로를 입력하세요.}'){
      sh 'npm install -g yarn'
      sh 'yarn --cwd ./ install --network-timeout 100000'
      sh 'yarn --cwd ./ build'
    }
  }
}

stage('Build') {
  steps {
    dir('{Dockerfile이 있는 경로를 입력하세요.}'){
      sh 'docker build -t {생성할 도커 이미지의 이름을 입력하세요} ./'
    }
  }
}

stage('Deploy') {
  steps{
    sh 'docker ps -f name={컨테이너 이름} -q | xargs --no-run-if-empty docker container stop'
    sh 'docker container ls -a -f name={컨테이너 이름} -q | xargs -r docker container rm'
    sh 'docker images --no-trunc --all --quiet --filter="dangling=true" | xargs --no-run-if-empty docker rmi'
    sh 'docker run -d --name {컨테이너 이름} -p 80:80 {생성한 도커 이미지의 이름}'
  }
}

stage('Finish') {
  steps{
    sh 'docker images -qf dangling=true | xargs -I{} docker rmi {}'
  }
}
}
}

```

11. Gitlab Webhook 설정

Search page

웹훅

웹훅을 사용하면 그룹 또는 프로젝트의 이벤트에 대한 응답으로 웹 애플리케이션에 알림을 보낼 수 있습니다. Webhook보다 통합을 사용하는 것이 좋습니다.

URL

URL must be percent-encoded if it contains one or more special characters.

Secret token

Used to validate received payloads. Sent with the request in the `X-Gitlab-Token` HTTP header.

Trigger

☒ Push events

Push to the repository.

☐ Tag push events

A new tag is pushed to the repository.

☐ 댓글

A comment is added to an issue or merge request.

☐ 비공개 댓글

A comment is added to a confidential issue.

☐ Issues events

An issue is created, updated, closed, or reopened.

☐ 비공개 이슈 이벤트

A confidential issue is created, updated, closed, or reopened.

☐ 머지 리퀘스트 이벤트

A merge request is created, updated, or merged.

☐ 작업 이벤트

A job's status changes.

☐ Pipeline events

A pipeline's status changes.

7. Jenkins 빌드 완료 화면

Dashboard >

+ 새로운 item

사람

빌드 기록

프로젝트 연관 관계

파일 빌거프린트 확인

Jenkins 관리

My Views

빌드 대기 목록

빌드 대기 항목이 없습니다.

빌드 실행 상태

1 대기 중

2 대기 중

상세 내용 입력

S	W	Name	이	최근 성공	최근 실패	최근 소요 시간	
✓	☀	apigateway-service		14 hr #29	10 days #1	27 sec	▶
✓	☀	battle-service		2 hr 59 min #178	3 days 7 hr #146	32 sec	▶
✓	☁	discovery-service		10 days #3	10 days #2	15 sec	▶
✓	☁	frontend		44 min #143	58 min #142	1 min 24 sec	▶
✓	☀	log-service		1 day 12 hr #41	3 days 8 hr #23	23 sec	▶
✓	☀	match-service		14 hr #25	6 days 14 hr #11	26 sec	▶
✓	☀	problem-service		1 hr 13 min #40	2 days 1 hr #22	25 sec	▶
✓	☀	rank-service		13 hr #37	6 days 10 hr #21	23 sec	▶
✓	☀	user-service		1 day 20 hr #53	—	24 sec	▶

아이콘: S M L

Icon legend

Atom feed 모두

Atom feed 실패

Atom feed 최근 빌드

8. QingDao OnlineJudge 설치

- 명령어를 통해 python pip curl git 설치

```
sudo apt-get update && sudo apt-get install -y vim python3-pip curl git
```

- 명령어를 통해 pip 업그레이드

```
pip3 install --upgrade pip
```

- 명령어를 통해 docker-compose 설치

```
pip install docker-compose
```

- 명령어를 통해 온라인저지 git pull

```
git clone -b 2.0 https://github.com/QingdaoU/OnlineJudgeDeploy.git && cd OnlineJudgeDeploy
```

- 해당 폴더로 접근 후 docker-compose.yml 파일에서 수정할 부분 수정 후 `docker-compose up -d` 명령어로 docker-compose 파일 실행

- 본 프로젝트에서는 port 번호만 수정하였음

9. Back-End DockerFile

- JAR 파일의 경로 관련 설정, java -jar /app.jar로 도커 container가 실행될 때 프로젝트가 동작하도록 설정

```
FROM openjdk:8
ARG JAR_FILE_PATH=build/libs/battle-service-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE_PATH} app.jar
EXPOSE 9002
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

10. Front-End DockerFile

- 작업 경로 설정 후 build 폴더 생성.
- jenkins를 통해 build한 내용을 작업 경로/build 폴더로 복사
- Nginx 설정 파일을 복사하고 포트 오픈, nginx 시작

```
## Dockerfile(client)

# nginx 이미지를 사용
```

```
FROM nginx

# work dir
WORKDIR /usr/share/nginx/html

# work dir 에 build 폴더 생성 : /home/test/client/build
RUN mkdir ./build

# host pc의 현재경로의 build 폴더를 workdir 의 build 폴더로 복사
ADD ./build ./build

# nginx 의 default.conf 를 삭제
RUN rm /etc/nginx/conf.d/default.conf

# host pc 의 nginx.conf 를 복사
COPY ./default.conf /etc/nginx/conf.d

# 80 포트 오픈
EXPOSE 82

# container 실행 시 자동으로 실행할 command. nginx 시작
CMD ["nginx", "-g", "daemon off;"]
```

11. Front-End Nginx Config 파일

- 80번 포트로 들어왔을 때, DockerFile에서 설정한 경로로 이동

```
server {
    listen 80;
    location / {
        root    /usr/share/nginx/html/build;
        index   index.html index.htm index.php;
        try_files $uri $uri/ /index.html;
    }
}
```

12. Port 정리

Port	Service
8000	apigateway-service
8761	discovery-service
9000	user-service
9001	problem-service
9002	battle-service
9003	rank-service
9005	log-service
9111	match-service
80	Front-End
1443	Judge Server