포팅 매뉴얼

목차

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

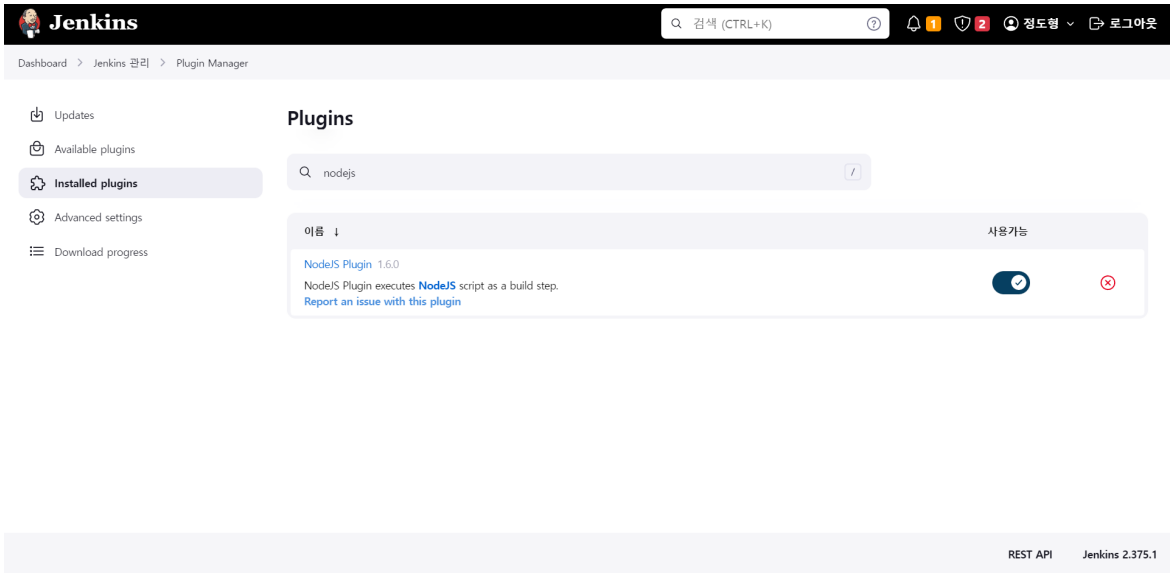
   `docker run --name mysql -e MYSQL_ROOT_PASSWORD={루트 패스워드 입력} -d -p 3306:3306 mysql`

6. Jenkins 설정

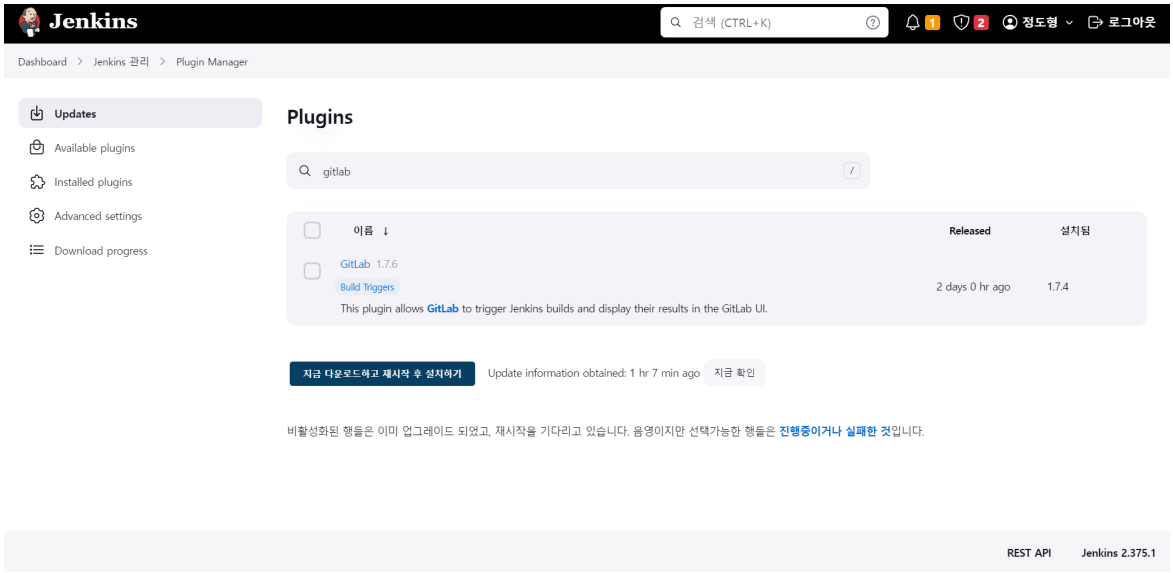
1. http://ip:6000/ 로 젠킨스 홈페이지 진입 후 패스워드창에 docker log jenkins 를 입력해서 얻어온 키값 입력

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

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

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

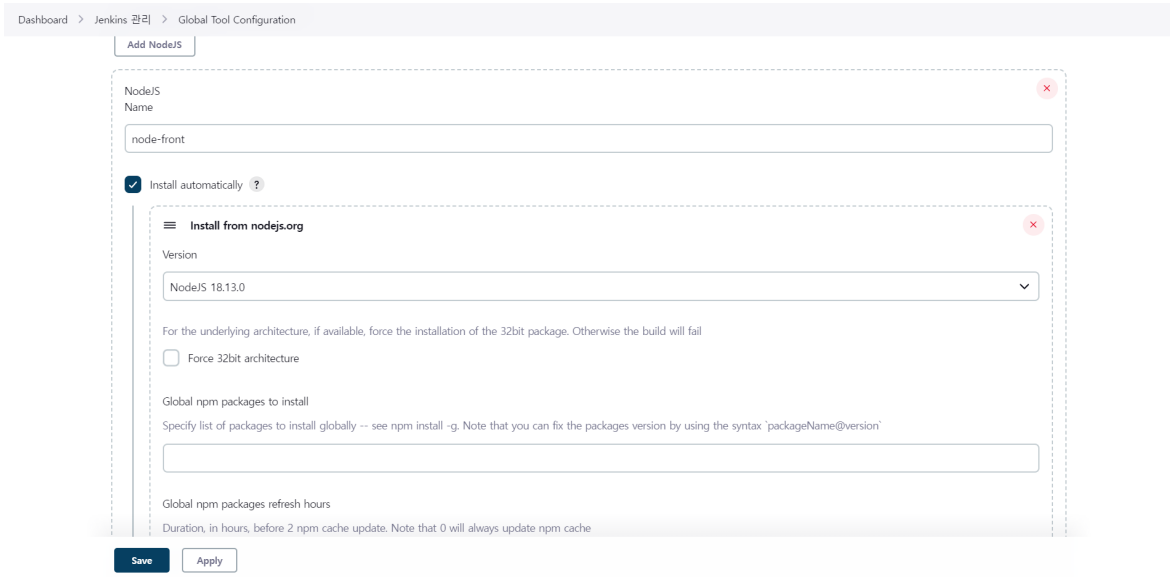
2. Jenkins 관리 → Plugin Manager에서 NodeJS Plugin 설치



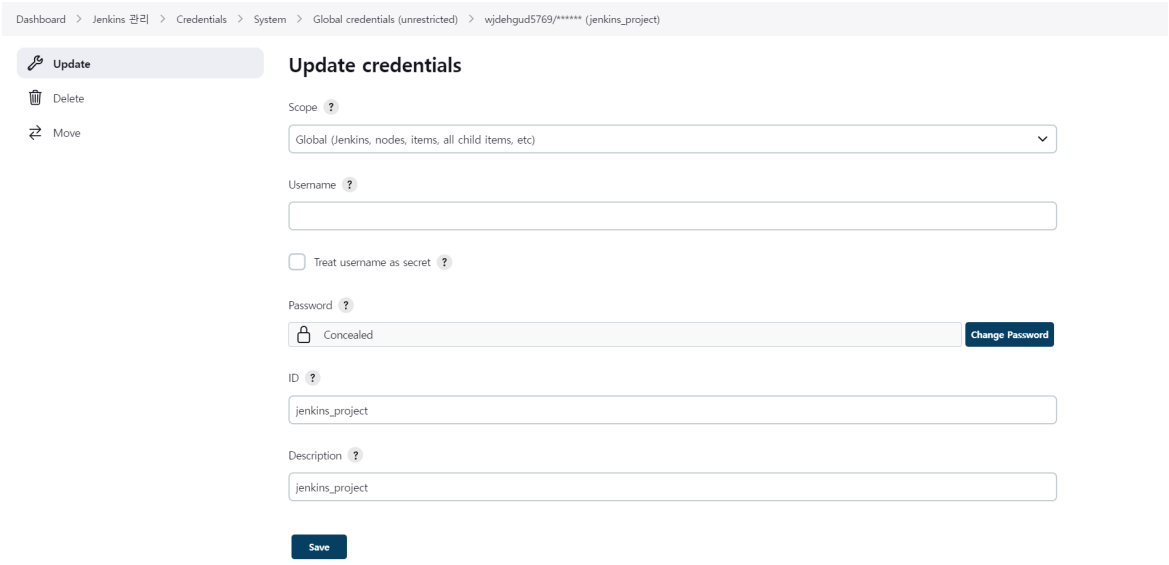
3. Jenkins 관리 → Plugin Manager에서 GitLab 관련 Plugin 설치



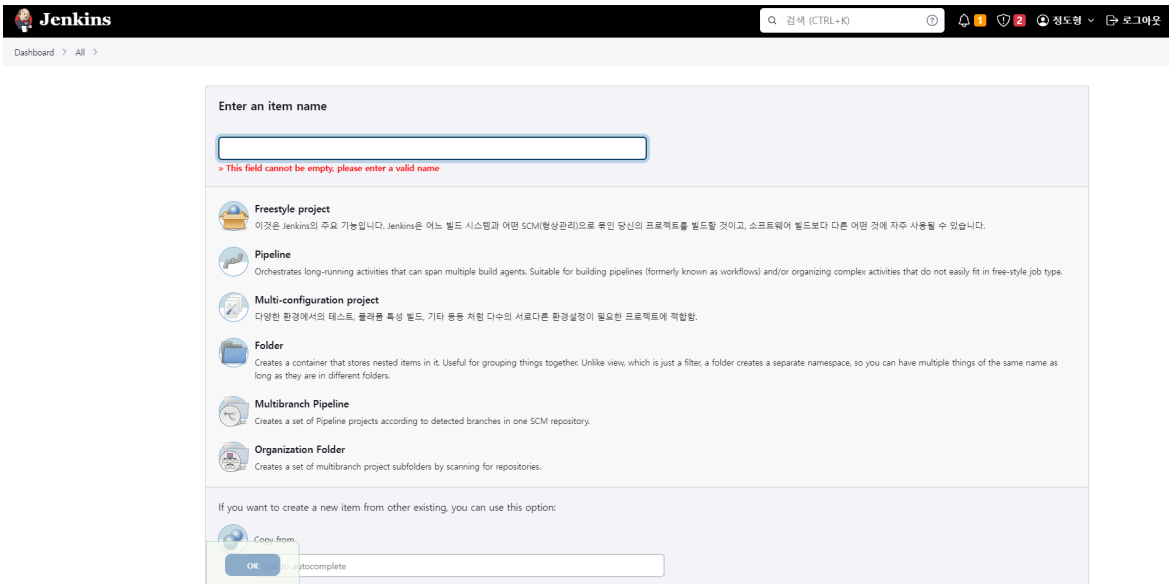
4. Jenkins 관리 → Global Tool Configuration에서 NodeJS 항목 추가



5. WebHook을 위한 Credential 설정
    - ID는 GItLab ID
    - Password는 Gitlab의 Project access Token



6. Jenkins 새로운 Item 만들기
   - jenkins 프로젝트 이름 입력 후 Pipeline으로 생성
   - Back-End 프로젝트 갯수만큼 Item 생성



7. Item GitLab 설정
   - Gitlab webhook 옵션 체크

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

8. 파이프라인 스크립트 입력(Back-End)

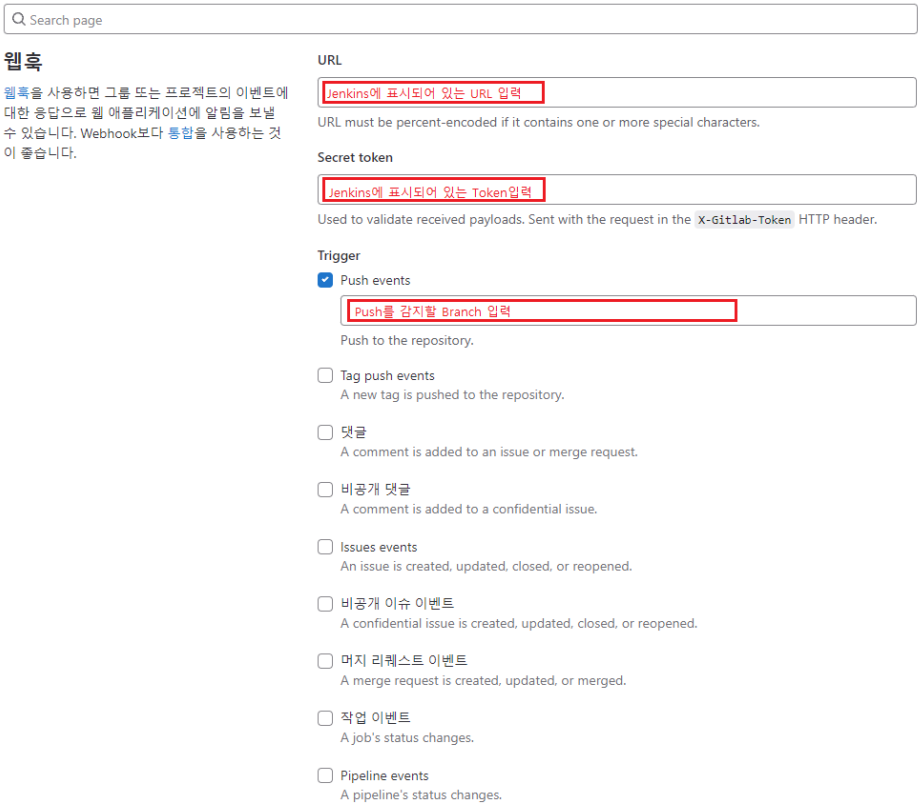
```
pipeline {
    agent any

    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 -fname={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)

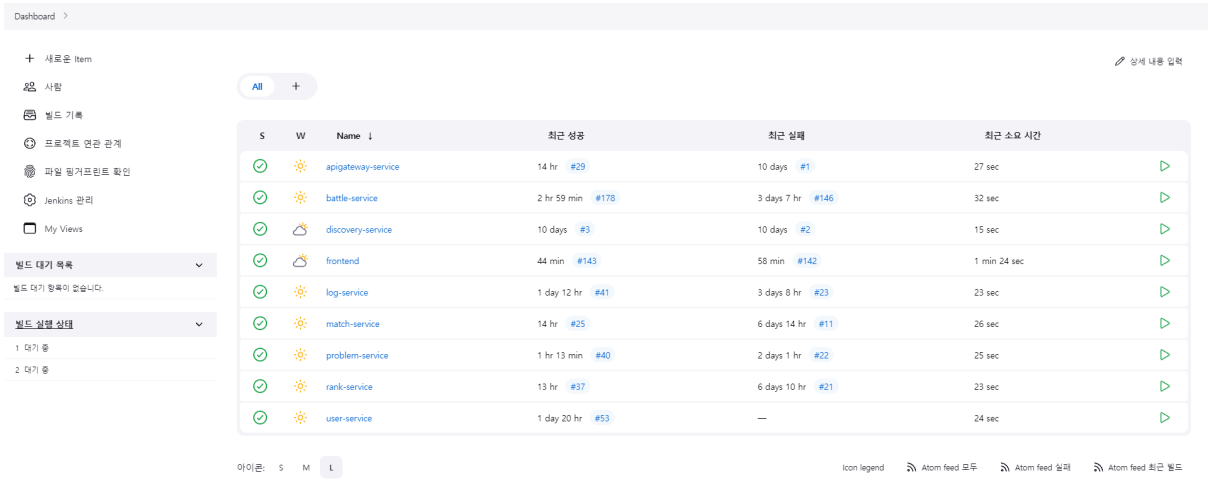
```
pipeline {
    agent any

    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 -fname={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



7. Jenkins 빌드 완료 화면



8. QingDao OnlineJudge 설치

1. 명령어를 통해 python pip curl git 설치

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

2. 명령어를 통해 pip 업그레이드

   `pip3 install –upgrade pip`

3. 명령어를 통해 docker-compose 설치

   `pip install docker-compose`

4. 명령어를 통해 온라인저지 git pull

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

5. 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 contianer가 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