

포팅 메뉴얼

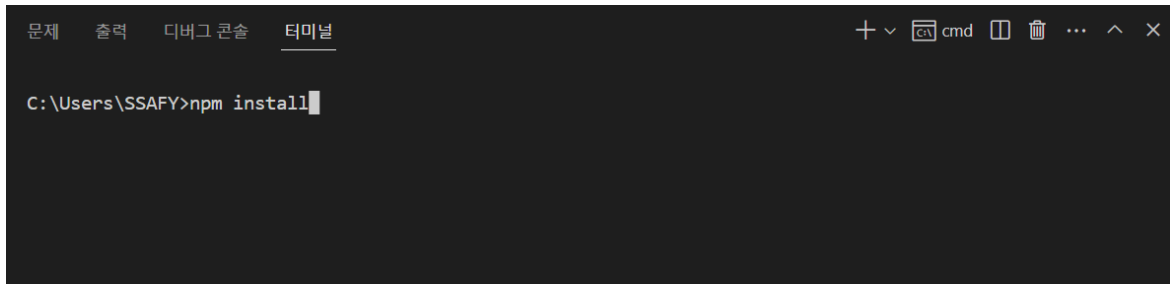
기술스택

1. 작업형상관리 : Jira
2. 형상관리 : Gitlab
3. 메신저 : Mattermost
4. 개발환경
 - a. OS : Window 10
 - b. IDE
 - i. Visual Studio Code
 - ii. IntelliJ IDEA
 - iii. MySQL Workbench : 8.0.3.1
 - c. DB
 - i. MySQL 8.0.31
 - ii. Redis 7.0.10
 - d. 프론트엔드
 - i. React 18.2.0
 - ii. Recoil 0.7.7
 - iii. Node.js 16.9.0
 - iv. node-sass 7.0.3
 - v. npm 7.21.1
 - e. 백엔드
 - i. SpringBoot 2.7.8
 - ii. Gradle 7.6.1
 - iii. JDK 11
 - iv. SpringSecurity
 - v. JPA
 - vi. Django 4.1.7
 - vii. Gunicorn 20.1.0
 - f. 서버
 - i. Amazon EC2
 - ii. Ubuntu 20.04.6 LTS
 - iii. Docker 23.0.1
 - iv. Jenkins 2.387.1
 - v. Nginx 1.18.0
 - vi. Amazon S3
 - vii. Docker-Compose 1.28.2

빌드 상세

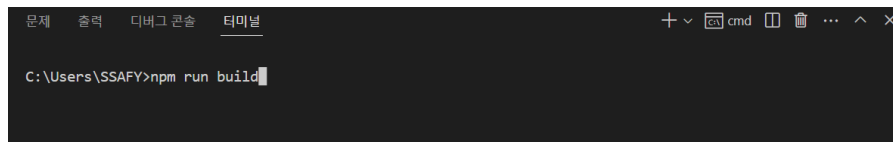
React

1. npm install 실행



```
C:\Users\SSAFY>npm install
```

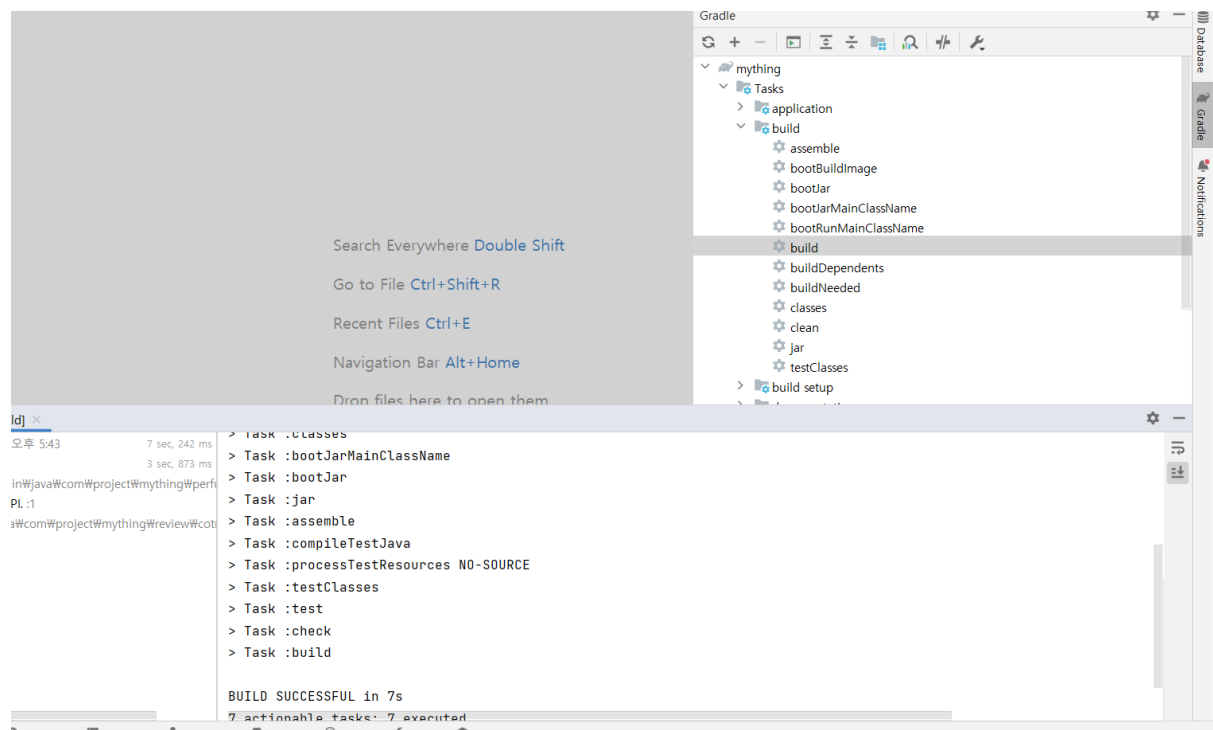
2. npm run build



```
C:\Users\SSAFY>npm run build
```

Spring-boot

1. Gradle 선택 후 task에서 build 실행



Django

1. `manage.py runserver` 실행

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\ssafy\rrrrr\S08P22B207\perfume> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 20 unapplied migration(s). Your project may not work properly until you apply them.
Run 'python manage.py migrate' to apply them.
April 05, 2023 - 20:19:02
Django version 4.1.7, using settings 'perfume.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

EC2 환경설정

• 도커 설치

모든 이미지, install 명령 실행 전 `sudo apt-get update`로 업데이트 이후 시작

```
1. 최신 패키지 리스트 업데이트
sudo apt update

2. docker 다운로드를 위해 필요한 https 관련 패키지 설치
sudo apt install apt-transport-https ca-certificates curl softwareproperties-common

3. docker repository 접근을 위한 GPG key 설정
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

4. docker repository 등록
sudo add-apt-repository "deb [arch=amd64]https://download.docker.com/linux/ubuntu focal stable"

5. 등록된 docker repository 까지 포함한 최신 패키지 리스트 업데이트
sudo apt update

6. docker 설치
sudo apt install docker-ce

서버 재부팅 할 때 도커를 자동으로 실행할 수 있는 설정
sudo systemctl enable docker.service
sudo systemctl enable containerd.service
```

• sudo 명령 없이 docker 명령어 사용하기 설정

```
1. 현 사용자(ubuntu)ID를 docker group 에 포함
sudo usermod -aG docker ${USER}

2. 터미널 끄고 다시 ssh로 접속 후 현 ID가 docker group에 포함되어 있는지를 확인
id -ng
```

• docker-compose 설치

```
1. docker-compose 설치
sudo curl -L "https://github.com/docker/compose/releases/download/1.28.2/dockercompose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

2. 실행 권한 주기
sudo chmod +x /usr/local/bin/docker-compose
```

• mysql 설치 및 연동

```
1. mysql 이미지 설치
docker pull mysql

2. mysql을 실행하기 위한 docker container 생성
docker run -d --name mysql-container -p 3300:3306 -e MYSQL_ROOT_PASSWORD=ns mysql

-d는 백그라운드 모드로 컨테이너를 실행하는 옵션
--name은 컨테이너의 이름을 정의
-p는 컨테이너의 포트와 호스트의 포트를 매핑 컨테이너의 3306포트를 호스트의 3300포트로 매핑
-e는 환경 변수를 설정하는 옵션 mysql의 root 패스워드를 'ns' 로 설정

3. 컨테이너 목록 출력
docker ps -a

4. mysql-container docker 컨테이너에서 bash 셸 실행
docker exec -it mysql-container bash

5. root 계정으로 접속
mysql -u root -p

6. 새로운 mysql 계정 생성
create user 'ns'@ '%' identified by 'ns';

ns 라는 이름의 계정을 생성하고 계정의 비밀번호를 ns로 설정
@ '%'는 어떤 호스트에서도 접근이 가능하도록 설정

7. mything 데이터베이스 생성
create database mything;

8. ns 계정에 데이터베이스에 대한 권한 부여
GRANT ALL PRIVILEGES ON mything.* TO 'ns'@ '%';

9. 변경사항 적용
FLUSH PRIVILEGES;

10. bash 셸 종료 및 컨테이너 종료
exit;
docker stop mysql-container
```

• redis 설치 및 연동

```
1. redis 이미지 설치
docker pull redis

2. redis을 실행하기 위한 docker container 생성
docker run -d --name redis-container -p 3301:6379 redis

-d는 백그라운드 모드로 컨테이너를 실행하는 옵션
--name은 컨테이너의 이름을 정의
-p는 컨테이너의 포트와 호스트의 포트를 매핑 컨테이너의 6379포트를 호스트의 3301포트로 매핑

3. redis cli 실행
docker exec -it redis-container redis-cli

4. 비밀번호 설정
CONFIG SET requirepass ns
```

• 젠킨스 설치

```
1. 젠킨스 컨테이너 생성
docker run -d -v /home/ubuntu/jenkins_home:/var/jenkins_home --name jenkins -p 8080:8080 -p 50000:50000 --restart=on-failure jenkins/jenkins:lts-jdk11

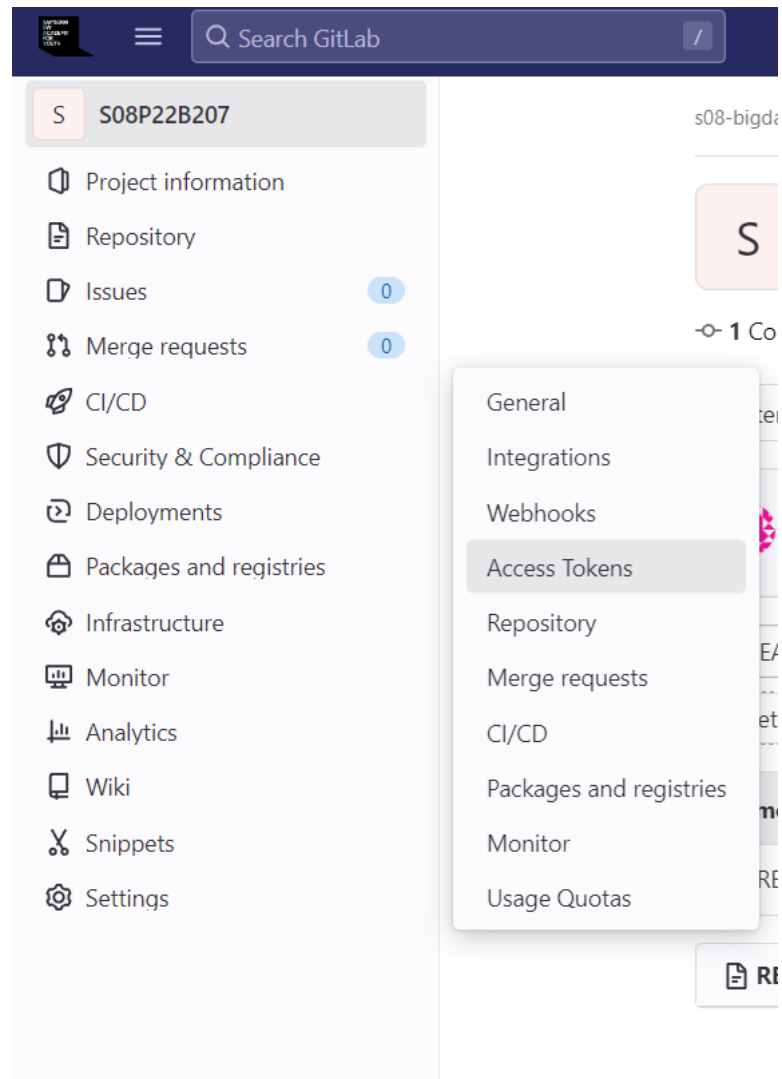
2-1. 초기 비밀번호 확인(로그확인)
docker logs jenkins

2-2. 초기 비밀번호 확인(컨테이너 접속)
docker exec -it -0 jenkins bash

cat /var/jenkins_home/secrets/initialAdminPassword
```

- Gitlab 연동

1. Access Token 발급



Project Access Tokens

Generate project access tokens scoped to this project for your applications that need access to the GitLab API.

You can also use project access tokens with Git to authenticate over HTTP(S). [Learn more.](#)

Add a project access token

Enter the name of your application, and we'll return a unique project access token.

Token name

token

For example, the application using the token or the purpose of the token. Do not give sensitive information for the name of the token, as it will be visible to all project members.

Expiration date

2023-04-22

Select a role

Maintainer

Select scopes

Scopes set the permission levels granted to the token. [Learn more.](#)

- ☒ api
Grants complete read and write access to the scoped project API, including the Package Registry.
- ☒ read_api
Grants read access to the scoped project API, including the Package Registry.
- ☒ read_repository
Grants read access (pull) to the repository.
- ☐ write_repository
Grants read and write access (pull and push) to the repository.

Create project access token

2. 젠킨스 설정

Jenkins 관리 > 시스템 설정 > Gitlab

Dashboard > Jenkins 관리 > Configure System >

Gitlab

☒ Enable authentication for '/project' end-point

GitLab connections

Connection name
A name for the connection

b207

Gitlab host URL
The complete URL to the Gitlab server (e.g. http://gitlab.mydomain.com)

https://lab.ssaify.com/s08-bigdata-recom-sub2/508P228207.git

Credentials
API Token for accessing Gitlab

GitLab API token

Add

고급

Test Connection

원하는 이름으로 설정, Gitlab host URL에는 깃랩 메인주소 입력 후 add 버튼 클릭

Jenkins Credentials Provider: Jenkins

Add Credentials

Domain: Global credentials (unrestricted)

Kind: GitLab API token

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

API token:

ID:

Description:

Add **Cancel**

깃랩에서 발급받은 Access Token을 입력하고 Add
Test Connection을 눌러 잘 연결 되었는지 확인
문제 없으면 Apply, 저장

3. 웹훅 설정

새로운 item을 클릭해서 Jenkins 프로젝트 생성

Enter an item name

b207

» Required field

- Freestyle project**
이것은 Jenkins의 주요 기능입니다. Jenkins은 어느 빌드 시스템과 어떤 SCM(형상관리)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 i type.
- Multi-configuration project**
다양한 환경에서의 테스트, 플러그인 특성 빌드, 기타 등등 처럼 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 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:

Copy from

Type to autocomplete

OK

Build Triggers(빌드 유발) 항목의 Build when a change is push to GitLab. GitLab webhook URL: 을 체크하고 고급 버튼 클릭
GitLab webhook URL 은 깃랩에서 연동할 때 필요

General

소스 코드 관리

빌드 유발

빌드 환경

Build Steps

빌드 후 조치

☐ Build periodically

☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://j8b207.p.ssafy.io:9090/project/b207>

Enabled GitLab triggers

☒ Push Events

☐ Push Events in case of branch delete

☒ Opened Merge Request Events

☐ Build only if new commits were pushed to Merge Request

☐ Accepted Merge Request Events

☐ Closed Merge Request Events

Rebuild open Merge Requests

Never

☒ Approved Merge Requests (EE-only)

☒ Comments

Comment (regex) for triggering a build

Jenkins please retry a build

고급

☐ Generic Webhook Trigger

☐ Gitlab Merge Requests Builder

☐ Poll SCM

고급 버튼 클릭 후 아래에 있는 Secret Token 생성 후 저장

Secret token

Generate

Clear

4. Gitlab 설정

webhook 설정 탭으로 이동해서 Gitlab webhook url을 입력하고 발급 받은 시크릿토큰도 입력

Webhooks

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an [integration](#) in preference to a webhook.

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**
Branch name or wildcard pattern to trigger on (leave blank for all)
Push to the repository.

☐ **Tag push events**
A new tag is pushed to the repository.

☐ **Comments**
A comment is added to an issue or merge request.

☐ **Confidential comments**
A comment is added to a confidential issue.

☐ **Issues events**
An issue is created, updated, closed, or reopened.

☐ **Confidential issues events**
A confidential issue is created, updated, closed, or reopened.

☐ **Merge request events**
A merge request is created, updated, or merged.

Trigger의 Push event에서 트리거를 발생시킬 브랜치의 이름을 입력
해당 브랜치에 푸시 이벤트가 일어날 때 마다 젠킨스에서 프로젝트가 빌드
Add webhook 버튼을 눌러서 완료

A feature flag is turned on or off.

☐ **Releases events**
A release is created or updated.

SSL verification

☒ **Enable SSL verification**

Add webhook

Project Hooks (3)

Push events

Tag push events

Issues events

Confidential issues events

Note events

Confidential note events

Merge requests events

Job events

Pipeline events

Test **Edit** **Delete**

Test **Edit** **Delete**

Test **Edit** **Delete**

푸시 이벤트 테스트도 가능

젠킨스 플러그인 설치 이슈

Deploy Plugin

You can select a plugin file from your local system or provide a URL to install a plugin from outside the central plugin repository.

파일



파일 선택

선택된 파일 없음

Or

사이트경로

Deploy

업데이트 사이트

사이트경로

Submit

업데이트 사이트 경로를 컨설턴트님이 아래 url로 바꿔 봄

<https://updates.jenkins.io/current/update-center.json>

>> 해결 안 됨 그냥 jenkins 홈페이지에서 플러그인 다운받아서 업로드 하는 방식으로 해결

젠킨스 컨테이너로 만들었기 때문에 빌드를 진행하기 위해 내부에 도커 설치

1. docker-compose.yml 파일로 한 번에 설치

```
1. 젠킨스 전용 디렉토리 생성
mkdir my-jenkins

2. 디렉토리 내부로 이동
cd /my-jenkins

3. Dockerfile 작성
vim Dockerfile

FROM jenkins/jenkins:lts

USER root

# install docker
RUN apt-get update && \
    apt-get -y install apt-transport-https \
        ca-certificates \
        curl \
        gnupg2 \
        zip \
        unzip \
        software-properties-common && \
    curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg > /tmp/dkey; apt-key add /tmp/dkey && \
    add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") \
    $(lsb_release -cs) \
    stable" && \
    apt-get update && \
```

```

apt-get -y install docker-ce

4. docker compose 파일 작성(8080포트로 젠킨스를 사용 안 하려면 젠킨스 포트는 변경)
vim docker-compose.yml

vim docker-compose.yml

version: '3'
services:
  jenkins:
    build:
      context: .
    container_name: jenkins
    user: root
    privileged: true
    ports:
      - 8080:8080
      - 50000:50000
    volumes:
      - ./jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock

5. jenkins 실행
docker-compose up -d

```

2. 젠킨스 컨테이너 생성 후 쉘에 접속하여 설치

```

1. jenkins 컨테이너 쉘에 접속
docker exec -it -u root jenkins bash

2. docker 설치
apt-get update && \
apt-get -y install apt-transport-https \
  ca-certificates \
  curl \
  gnupg2 \
  software-properties-common && \
curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg > /tmp/dkey; apt-key add /tmp/dkey && \
add-apt-repository \
  "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") \
  $(lsb_release -cs) \
  stable" && \
apt-get update && \
apt-get -y install docker-ce

3. Docker그룹에 root 계정 추가
usermod -aG docker root
su - root

4. 그룹에 잘 추가되었는지 확인
id -nG // "root docker" 가 뜨면 정상

5. docker.sock 권한 변경
chmod 666 /var/run/docker.sock

6. root에서 도커 로그인
su - root
docker login

```

추가로 Spring은 Gradle로 빌드하기때문에 별도의 패키지 설치가 필요 없지만

React같은 경우 node.js로 빌드하기 때문에 젠킨스 컨테이너 내부에 node와 npm 설치를 해야함

spring-boot 배포 과정

1. Dockerfile 작성

```
1. openjdk:11-slim 이미지를 사용해서 빌드
FROM openjdk:11-slim

2. /tmp 경로로 볼륨
VOLUME /tmp

3. 8080포트 오픈
EXPOSE 8080

4. jar 파일 생성
ARG JAR_FILE=build/libs/mything-0.0.1-SNAPSHOT.jar

5. /tmp 경로로 jar파일 복사
COPY ${JAR_FILE} Mything.jar

ENTRYPOINT [ "java", "-jar", "/Mything.jar" ]
ARG DEBIAN_FRONTEND=noninteractive
ENV TZ=Asia/Seoul
RUN apt-get install -y tzdata
```

2. 빌드 과정 설정(파이프라인 스크립트 작성과 동일한 과정)

Dashboard > 젠킨스 아이템 선택 > 구성

Build Steps

≡

Invoke Gradle script ?

✕

☐ Invoke Gradle ?

☒ Use Gradle Wrapper ?

☒ Make gradlew executable

Wrapper location ?

Tasks ?

▼

🔍

≡

Execute shell ?

✕

Command

See [the list of available environment variables](#)

```
cd /var/jenkins_home/workspace/release-BE/mything
./gradlew clean build

docker build -t ehdrnfd1002/b207 .
docker push ehdrnfd1002/b207
```

🔍

Invoke Gradle script는 Jenkins에서 Gradle 스크립트를 실행하는 단계

Gradle 스크립트를 실행하고, 빌드 결과를 출력합니다. 이 단계에서는 Gradle을 설치하고, 프로젝트 빌드에 필요한 의존성을 다운로드합니다. 그리고 빌드 스크립트를 실행하여 빌드를 수행하고, 빌드 결과를 출력

Wrapper location은 Gitlab에서 프로젝트 디렉토리로

release-BE

S08P22B207 / +

History

Find file

Web IDE

741334a4

Clone

도커파일추가

최동호 authored 3 hours ago

741334a4

Name	Last commit	Last update
Servers	도커파일추가	3 hours ago
mything	도커파일추가	3 hours ago
.gitignore	도커파일추가	3 hours ago
README.md	add README	2 weeks ago

\${workspace}는 release-BE까지의 경로가 되므로 \${workspace}/mything으로 작성해주면 됨

Execute shell은 쉘 명령어를 실행 하는 과정으로 빌드가 완료되고 jar 파일이 생성된 이 후 실행

```

1. Dockerfile 위치로 이동
cd /var/jenkins_home/workspace/release-BE/mything

2. 이전에 빌드 된 결과를 삭제하고 다시 빌드
./gradlew clean build

3. 이미지 빌드 > 허브에 이미지를 업로드하고 받는 과정 할 필요 없이 그냥 바로 이미지 빌드 실행
docker build -t spring .

4. 기존에 생성한 컨테이너 중지 후 제거
docker ps -q --filter name=spring | grep -q . && docker rm -f $(docker ps -aq --filter name=spring)

5. 8080 포트로 빌드한 이미지로 컨테이너 다시 생성
docker run -d --name spring -p 8080:8080 spring:latest

6. proxy 컨테이너를 docker-compose로 생성해서 proxy 컨테이너는 nginx_default 네트워크에 속하므로 같은 네트워크에 추가
docker network connect nginx_default spring

7. 이전에 사용했던 이미지를 제거
docker image prune -f --filter "dangling=true"
```

도커 허브를 사용할 경우 허브와 ec2를 연결하기 위한 과정

ssh 서버 접속 설정

jenkins 관리 > 시스템 설정 > Pushlish over SSH(ssh 플러그인 설치)

Publish over SSH

Jenkins SSH Key ?

Passphrase ?

Concealed

Change Password

Path to key ?

Key ?

```
-----BEGIN RSA PRIVATE KEY-----
MIIEEwIBAAKCAQEAPEXwhr9ITA2HuZhoDg8sIOKJeriO0RdeK2q3ZYomERnw
OTDcOX1w+e2Bzpvvg1HSahbVjmsUKpZYLmdXytaGLgKlWLTcUeZlgAnFpZTJ
3qbKshUF8yUTLUDPBmcf1Em7kbeVbafq35rPKhzF8igzoVf0t+/hepllcQeElwn
9CF04EceOp2fntNj5n8dNQAWksL+/y66b1yQvQzAUlImhuDrFaklg/P81Z4K/7hy
qXhiwnX5SmOOfhLxC9Pl/pbhrXRKgtt3NYVWZy+MMe3lq5pJsoM74V08mORhBKHC
JmmQpMLUWqW1xYd0mWQB9hXSHBDVlrlep4AcwIDAQABAgQIBACiFcbF45KBCO+z
TQyH9R1TA2Vib0+ofhctCySiaZQr+KOLjmh952f8QZSLJLHkKu3h7AmVSVG6bc2l
EAp02tkFk3H3ufw34hWj5e9MMXly9eH7JYawc4ewB4wcXvoL0voDwnXKsKJ6O
8pxCnUE5nCSdn39Pye0SqwDrMuJZx64T9IMXOYnfgKd7d7zOEjhZPWS3/xye
ZZQuriAeg3r/Ks87mo1hCfrEYURgJOcK9Eh+G51XLkIKPkE7LRsU00b8yacF
SISxLV8cJV1bwzflNw54d/6m/KULCwVFBio33bt9VBdhcQUJ1DQDrJ5008m5esP
Yq2TBZECgyEAmn9agePjzck7mhfx8BkAfYdYogY3QfR/oLZaZcaeeyzqVdZQooF
FKPCuDOXKj89dZuWLDUJmLX0CAVE1toja841EnZhfFgWp+4ZB61S/l+OCbRxbZjw
AvTYR25F7f4N/qzpcckA/VbLkicPipkYkEs2T53GQqbeZ/56jX8UCyEAuZM
d/vXaU8oTnEhDr85y3FjXgKj0eL3BEoBtoLipolblzV0+wmLSqaPZLguOKUEIO
w3tNhOSWgKu5qV2N+qojsyinUwOUIRTwMXecabgOeMeCCKGUck7UEz1evZHBIESW
uXMXQs77A/L6FuERxoXoepD2mfADPjEKHBGc16tcCgYEAryDSJFCJDOMlbt/4+Tq8
URuXP7/59KqUOIOWyRJS8qeFpB2gS8K8RFoSuVs5zt4tUWcthNVKcmFejyDOei
e5CCb0C+J16WR/YnBT8Wx8K5/mwT+5vclMmL49QP0p4ATA8RCUmkbui03fsYufk
+QVP3QIDAh7MLF2GsX30CgyB55A8eNUsU9RC+KSDXpm5Y5DErMPBIMU1DFq
4SweixEMCzwR42a3hskh2wz20XgY2eHdUa5lpa5CWB28CCzLJMG2282D2GQeOtnVTU
3zdY2FhFhduorgvltz1mY/sYv/b08TOM3rMBjlt6VOw+vQahKNF5sO1GuVvz
9wOoSwkBgGEdpZgfpC78kpDNmh2bcODr2NIMshhwDfCn36wAM8HJ/DwUSGlnPeJ
EjNNHBMplcEmfVdQt18yp7CzTvcL8yn89g+ybpyVw4lmt8+8hLv8jXGq8WU6ln
qIsTugleqXlccmn+8DuJISWcELuvhLrX/69AZOTukJX5bgsUW
-----END RSA PRIVATE KEY-----
```

key에 pem키 붙여넣기

SSH Servers

SSH Server

Name ?

ec2

Hostname ?

j8b207.p.ssafy.io

Username ?

ubuntu

Remote Directory ?

고급

Test Configuration

Name : 젠킨스 내에서 사용할 이름

Hostname : 서버 도메인

Username : 유저이름

다 작성 후 Test Configuration 실행 > Success가 나오면 성공

다시 Dashboard > 젠킨스 아이템 선택 > 구성

☒ Send files or execute commands over SSH after the build runs ?

SSH Publishers

SSH Server

Name ?

ec2

고급 ▾

Transfers

Transfer Set

Source files ?

Remove prefix ?

Remote directory ?

Exec command ?

```

docker login -u ehdrmfldl002 -p cgw!951002 docker.io
docker pull ehdrmfldl002/b207:latest
docker ps -q --filter name=spring | grep -q . && docker rm -f $(docker ps -aq --filter name=spring)
docker run -d --name spring -p 8080:8080 ehdrmfldl002/b207:latest

```

All of the transfer fields (except for Exec timeout) support substitution of [Jenkins environment variables](#)

고급 ▾

저장 Apply

Send files or execute commands over SSH after the build runs는 빌드가 완료된 후 SSH를 통해 원격 서버로 파일을 전송하거나 명령을 실행할 수 있는 옵션

빌드된 아티팩트를 원격 서버로 전송하거나, 원격 서버에서 특정 명령을 실행하여 애플리케이션을 배포하거나, 원격 서버에서 파일을 삭제하는 등의 작업을 수행

Exec command 작성

```

1. 도커허브 로그인
docker login -u ehdrmfldl002 -p cgw!951002 docker.io

2. 도커 허브에서 최신 이미지 pull
docker pull ehdrmfldl002/b207:latest

3. name으로 생성된 기존의 컨테이너를 중지 후 삭제
docker ps -q --filter name=spring | grep -q . && docker rm -f $(docker ps -aq --filter name=spring)

4. pull 받은 최신 이미지로 name으로 컨테이너 생성
docker run -d --name spring -p 8080:8080 ehdrmfldl002/b207:latest

```

React 배포 과정

젠킨스와 깃랩 연결은 위와 동일

1. 도커 파일 작성

React는 Spring처럼 자체 서버가 없기 때문에 node.js로 빌드

그 다음 nginx를 사용해서 웹 서버를 관리하기 위해 node.js로 빌드 된 결과물을 nginx를 사용해 도커 이미지를 생성

```
# base image
FROM node:16.9.0-alpine AS build

# set working directory
WORKDIR /app

# install dependencies
COPY package*.json ./
RUN npm install

# build app
COPY . .
RUN npm run build

FROM nginx:alpine

# copy custom Nginx config
COPY nginx.conf /etc/nginx/conf.d/default.conf

# set working directory
WORKDIR /app

# copy build files from build container
COPY --from=build /app/build /app

# expose port
EXPOSE 80

# start Nginx server
CMD ["nginx", "-g", "daemon off;"]
```

2. nginx 설정

nginx를 사용해 웹 서버를 열었기 때문에 외부에서 포트에 접근하기 위해 nginx.conf 파일 수정

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

3. 파이프 라인 쉘 스크립트

위와 동일하게 도커파일이 존재하는 위치로 이동 후 빌드 과정 작성

```
cd /var/jenkins_home/workspace/release-FE/front

docker build --no-cache -t react .
docker ps -q --filter name=react | grep -q . && docker rm -f $(docker ps -aq --filter name=react)
docker run -d --name react -p 3000:80 react:latest
docker network connect nginx_default react
docker image prune -f --filter "dangling=true"
```

Django 배포 과정

젠킨스 설정은 위와 동일

1. 도커 파일 작성

- 배포 환경에선 Gunicorn 서버를 사용하는 이유
 - Gunicorn은 Python WSGI로 웹 서버(Nginx)로부터 서버사이드 요청을 받으면 WSGI를 통해 서버 어플리케이션으로 전달해주는 역할
 - Django의 "runserver" 또한 같은 역할을 수행하지만 보안, 성능적으로 production 환경에서는 적절하지 않음
 - WSGI는 멀티 쓰레드를 만들 수 있기 때문에 Request가 많아지더라도 효율적으로 처리 가능
- pip freeze > requirements.txt을 사용해서 install 해야되는 패키지들을 txt 문서로 만들고 이 때 Gunicorn도 포함되게 함
- "perfume.wsgi:application"의 경로는 wsgi:application 파일이 위치하는 경로로 작성

```
FROM python:3.9.13

WORKDIR /app

COPY requirements.txt .
RUN pip install --upgrade pip
RUN pip install -r requirements.txt

COPY . .

CMD ["gunicorn", "--bind", "0.0.0.0:8000", "perfume.wsgi:application"]
```

2. 파이프라인 쉘 스크립트

위와 동일하게 도커파일이 존재하는 위치로 이동 후 빌드 과정 작성

```
cd /var/jenkins_home/workspace/release-BIGDATA/perfume

docker build -t django .
docker ps -q --filter name=django | grep -q . && docker rm -f $(docker ps -aq --filter name=django)
docker run -d --name django -p 8081:8000 django:latest
docker network connect nginx_default django
docker image prune -f --filter "dangling=true"
```

SSL 설정 및 proxy 컨테이너 생성

• certbot 과 nginx 기본 설정

```
certbot certonly --dry-run --webroot --webroot-path=/usr/share/nginx/html --email test@test.com --agree-tos --no-eff-email --keep-until-expiring -d j8b207.p.ssafy.io -d j8b207.p.ssafy.io
```

- certbot certonly : certonly 는 certbot 프로그램의 플러그인으로 인증서를 받는 서버명령
- certbot/certbot 이미지는 entrypoint 가 certbot 이기 때문에, command 에서는 certonly 부터 작성함
- certonly 는 새로 인증서를 받거나, 갱신된 인증서를 받기만 하며, certonly 와 --webroot 옵션을 함 께 써서, certbot 프로그램이 알맞은 위치에 인증서를 설치하게 됨
- --dry-run : 테스트시에는 반드시 이 옵션을 써서 테스트, 이 옵션 없이 일정 기간동안 수차례 테스트하면, 에러 발생
- --webroot : 내 서버에 인증 정보를 넣고, 이를 기반으로 인증서를 발급받겠다는 옵션 (인증서 갱신을 위해, 웹서버를 끄지 않아도 됨)
- --webroot-path : 인증 정보를 넣을 내 서버의 기본 폴더 지정
- --email : 인증서 복구등을 위한 이메일 등록
- --agree-tos : ACME 서버 구독 동의

- --no-eff-email : 해당 이메일 주소를 certbot 회사에 공유하지는 않음
- --keep-until-expiring : 아직 인증서가 renew 가 필요하지 않으면, 갱신하지 않고 기존 인증서를 보존함

• docker-compose.yml

```
version: "3"

services:
  webservers:
    image: nginx:latest
    container_name: proxy
    restart: always
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf
      - ./certbot-etc:/etc/letsencrypt

  certbot:
    depends_on:
      - webservers
    image: certbot/certbot
    container_name: certbot
    volumes:
      - ./certbot-etc:/etc/letsencrypt
      - ./myweb:/usr/share/nginx/html
    command: certonly --dry-run --webroot --webroot-path=/usr/share/nginx/html --email [test@test.com](mailto:test@test.com) --agree-tos --no-eff-email --keep-until-expiring -d j8b207.p.ssafy.io
```

• nginx.conf 설정

- /.well-known/acme-challenge location 설정은 certbot 의 --webroot 옵션을 통해, 인증서 발급을 위 해 해당 location 을 접근하므로, 필요함
- 우선 인증서 발급 및 <http://j8b207.p.ssafy.io>로 nginx proxy 및 인증서 발급 정상 동작까지만 확인

• nginx.conf

```
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log warn;
pid /var/run/nginx.pid;
events {
    worker_connections 1024;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
        '$request_uri' "$uri"
        '$status $body_bytes_sent "$http_referer"
        "$http_user_agent"
        "$http_x_forwarded_for"';
    access_log /var/log/nginx/access.log main;
    sendfile on;
    keepalive_timeout 65;
    upstream docker-web {
        server nginx:80;
    }
    server {
        location ~ /.well-known/acme-challenge {
            allow all;
            root /usr/share/nginx/html;
            try_files $uri =404;
        }
        location / {
            allow all;
        }
    }
}
```

```

    root /usr/share/nginx/html;
    try_files $uri =404;
  }
}
}

```

- 인증서 발급 확인

- docker-compose up -d 명령 후, certbot 컨테이너 로그 확인

```

ubuntu@ip-172-31-33-93:~/08_HTTPS$ docker logs certbot
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Plugins selected: Authenticator webroot, Installer None
Requesting a certificate for fun-coding.xyz and www.fun-coding.xyz
Performing the following challenges:
http-01 challenge for fun-coding.xyz
http-01 challenge for www.fun-coding.xyz
Using the webroot path /usr/share/nginx/html for all unmatched
domains.
Waiting for verification...
Cleaning up challenges
IMPORTANT NOTES:
- dry run was successful!

```

- 인증서 파일 확인 (dry-run 옵션 삭제 후, 실제 인증서 발급시)

```

# EC2 root 사용자 패스워드 설정
sudo passwd

# root 사용자로 사용자 전환
su -

cd /home/ubuntu/08_HTTPS/certbot-etc/live/
ls

#자신의 도메인 폴더가 있고, 해당 폴더 안에 인증서가 있음

```

- 보안 옵션 개선

- 호스트 PC에서 docker compose 에 들어갈 volume 에 직접 다음 두 파일을 정확한 위치에 다운로드 받음 curl 다운로드 및 설치 (<https://curl.se/download.html> 또는 <https://winampplugins.co.uk/curl/>)
 - options-ssl-nginx.conf : https://raw.githubusercontent.com/certbot/certbot/master/certbot/nginx/certbot/_internal/tls_configs/options-ssl-nginx.conf
 - ssl-dhparams.pem : <https://raw.githubusercontent.com/certbot/certbot/master/certbot/certbot/ssl-dhparams.pem>

```

cd certbot-etc

# options-ssl-nginx.conf 다운로드
curl -s [https://raw.githubusercontent.com/certbot/certbot/master/certbot-](https://raw.githubusercontent.com/certbot/certbot/master/c

# ssl-dhparams.pem 다운로드
curl -s [https://raw.githubusercontent.com/certbot/certbot/master/certbot/c](https://raw.githubusercontent.com/certbot/certbot/master/

# 즉, 컨테이너 내에 /etc/letsencrypt 폴더에 위 두 파일이 존재하도록 하면 됨

```

- nginx.conf 개선

http:// 로 접속시에는 https:// 로 redirection
https:// 로 접속 지원을 위해, 443 포트 오픈 및 ssl 프로토콜 지원을 위한 보안 키 설정

proxy 패스 설정 과정

```
upstream spring-api {
    server spring:8080;
}
```

- 도커라이징 된 컨테이너로 들어가기 위해 호출 url을 설정하기 위해 컨테이너 호스트네임과 내부 포트를 지정해줌
- spring가 컨테이너명, 8080이 spring컨테이너 내부에서 오픈 된 포트
- spring-api은 url 호출하기 위한 메소드명
- 엔드포인트를 구분하기 위해 location에 엔드포인트 명을 구분해서 작성한 뒤
- 프록시패스에서 구분하기 위한 엔드포인트 뒷 부분을 잘라서 컨테이너로 보내기 위해rewrite ^/api(.*)\$ \$1 break; 코드를 사용

http://j8b207.p.ssafy.io/api/join 로 Reat api 요청이 들어 온 경우

- http://j8b207.p.ssafy.io/ 은 80포트로 들어 온 요청이라

```
location / {
    return 301 https://$host$request_uri;
}
```

에서 443포트(https) 요청으로 보내줌

이후 rewrite ^/api(.*)\$ \$1 break; 에서 api 이후에 작성된 join을 잘라서

http://j8b207.p.ssafy.io:8080/join 으로 요청이 들어감

```
user nginx;
worker_processes auto;

error_log /var/log/nginx/error.log warn;
pid /var/run/nginx.pid;

events {
    worker_connections 1024;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;
    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
        '$status $body_bytes_sent "$http_referer" "$request_uri" "$uri"'
        '"$http_user_agent" "$http_x_forwarded_for"';
    access_log /var/log/nginx/access.log main;
    sendfile on;
    keepalive_timeout 65;

    upstream react-web {
        server react:80;
    }

    upstream spring-api {
        server spring:8080;
    }

    upstream django-api {
        server django:8000;
    }

    server {
        listen 80;
        listen [::]:80;
        server_name j8b207.p.ssafy.io;
        client_max_body_size 10M;

        location / {
            return 301 https://$host$request_uri;
        }
    }
}
```

```

server {
    listen 443 ssl;
    server_name j8b207.p.ssafy.io;
    client_max_body_size 10M;

    ssl_certificate /etc/letsencrypt/live/j8b207.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/j8b207.p.ssafy.io/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location / {
        proxy_pass          http://react-web;
        proxy_redirect      off;
        proxy_set_header    Host $host;
        proxy_set_header    X-Real-IP $remote_addr;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header    X-Forwarded-Host $server_name;
        proxy_set_header    X-Forwarded-Proto $scheme;
    }

    location /api/ {
        rewrite ^/api(.*)$ $1 break;
        proxy_pass          http://spring-api;
        proxy_redirect      off;
        proxy_set_header    Host $host;
        proxy_set_header    X-Real-IP $remote_addr;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header    X-Forwarded-Host $server_name;
        proxy_set_header    X-Forwarded-Proto $scheme;
    }

    location /bigdata/ {
        rewrite ^/bigdata(.*)$ $1 break;
        proxy_pass          http://django-api;
        proxy_redirect      off;
        proxy_set_header    Host $host;
        proxy_set_header    X-Real-IP $remote_addr;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header    X-Forwarded-Host $server_name;
        proxy_set_header    X-Forwarded-Proto $scheme;
    }

    location /find-image {
        rewrite ^/find-image(.*)$ $1 break;
        proxy_pass          http://image:80;
        proxy_redirect      off;
        proxy_set_header    Host $host;
        proxy_set_header    X-Real-IP $remote_addr;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header    X-Forwarded-Host $server_name;
        proxy_set_header    X-Forwarded-Proto $scheme;
    }
}

```

- 이미지 서버 생성

```

docker run -d --name image --restart always \
--network nginx_default \
-p 8082:80 \
-v /home/ubuntu/images/perfume_image:/usr/share/nginx/html/images/perfume_image \
-v /home/ubuntu/images/note_image:/usr/share/nginx/html/images/note_image \
-v /home/ubuntu/images/user_image:/usr/share/nginx/html/images/user_image \
-v /home/ubuntu/images/review_image:/usr/share/nginx/html/images/review_image \
-v /home/ubuntu/images/survey_image:/usr/share/nginx/html/images/survey_image \
-v /home/ubuntu/images/guide_image:/usr/share/nginx/html/images/guide_image \
-v /home/ubuntu/images/note_class:/usr/share/nginx/html/images/note_class \
-v /home/ubuntu/images/main_image:/usr/share/nginx/html/images/main_image \
nginx:latest

```

- ec2 내부에 디렉토리를 생성해 디렉토리에 이미지들을 저장하고 nginx 이미지, nginx의 root 경로에 volume을 설정