포팅 매뉴얼

기술 스택

이슈 관리

Jira

형상 관리

Gitlab

커뮤니케이션

Mattermost, Notion, Figma

개발 환경

1. OS

- Windows 10

2. Web server
   - NGINX 1.18.0
3. WAS
   - Apache Tomcat 9.0.65
4. IDE
   - IntelliJ IDEA 2022.1.4
   - Visual Studio Code 1.70.0
5. DB
   - MySQL 8.0.30
   - Redis 5.0.7
6. 배포
   - AWS EC2 Ubuntu 20.04.5 LTS
   - Docker 20.10.18
   - Jenkins 2.368

상세 사용

1. Backend
   - Java 17.0.3.1
   - Spring Boot 2.7.3
   - Spring Data JPA 2.7.3
   - Spring Security 5.7.3
   - Gradle 7.5
   - lombok 1.18.24
2. Frontend
   - React 18.2.0
   - TypeScript 4.8.2
   - React-Router-Dom v6 6.3.0
   - Redux 4.2.0
   - Axios 0.27.2
   - Tailwind CSS 3.1.8
   - Chart.js 3.9.1

3. Data
   - EC2 환경 기준
     - Spark 3.2.2
       - Scala 2.12.15, OpenJDK 64-Bit Server VM, 1.8.0_342
     - Zeppelin 0.10.1
     - python(colab)

AWS EC2 ubuntu 서버 세팅

※ root 계정 로그인

1. root 계정 전환 명령어

```
su root
```

2. 비밀번호 입력

Docker

- 설치
  - 사전 패키지 다운로드

```
# apt 패키지 업데이트
apt update

# 사전 패키지 설치
apt-get install -y ca-certificates \
                   curl \
                   software-properties-common \
                   apt-transport-https \
                   gnupg \
                   lsb-release

# 위의 명령어로 패키지 설치가 잘 안된다면 각각 나눠서 설치해도 됨
apt-get install -y ca-certificates
apt-get install -y curl
apt-get install -y software-properties-common
apt-get install -y apt-transport-https
apt-get install -y gnupg
apt-get install -y lsb-release
```

  - gpg 키 다운로드

    리눅스 패키지 툴이 프로그램 패키지가 유효한지 확인하기 위해, 프로그램 설치 전에 gpg 키로 검증하는 과정을 거치기 때문에 gpg 키를 받아야 함

```
mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo \
    "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

  - 도커 설치

```
apt update
apt install docker-ce docker-ce-cli containerd.io docker-compose
```

    위의 명령어로 설치하는 과정에서 문제가 발생할 경우 다음 명령어들을 순서대로 수행 후 재시도

```
# 패키지 리스트 업데이트
sudo apt update

# apt가 https를 통해 패키지를 사용하기 위한 선행 패키지들을 설치
sudo apt install apt-transport-https ca-certificates curl software-properties-common

# 시스템 상에 공식 도커 저장소 접속을 위한 GPG 키를 추가
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

# 도커 저장소를 apt 소스에 추가
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docke

# 추가된 내용을 적용하기 위해 다시 패키지 리스트 업데이트
sudo apt update

# ubuntu 기본 저장소가 아닌 도커 저장소를 사용하도록 설정
apt-cache policy docker-ce
```

- 관련 파일
  - `Dockerfile`(Backend용)

    ▼ 코드 보기

    ```
    # 빌드를 위해 gradle을 사용
    FROM gradle:7.5

    # 작업 디렉토리를 고정
    WORKDIR /var/jenkins_home/workspace/gamulgamul/backend

    # 호스트의 현재 폴더에 있는 내용을 컨테이너에 복사
    COPY ./ ./

    # gradle의 clean, build 명령어를 순서대로 수행
    RUN gradle clean build --no-daemon

    # 컨테이너 실행시 `java -jar ./build/libs/gamulgamul-0.0.1-SNAPSHOT.jar`를 수행
    CMD ["java", "-jar", "./build/libs/gamulgamul-0.0.1-SNAPSHOT.jar"]
    ```

  - `Dockerfile`(Frontend용)

    ▼ 코드 보기

    ```
    # 빌드를 위해 node를 사용
    # Multi Stage Build 방식을 활용하기 위해 build-stage라는 이름의 스테이지로 정의
    FROM node:16.15.0 as build-stage

    # 작업 디렉토리를 고정
    WORKDIR /var/jenkins_home/workspace/frontend/frontend

    # 현재 디렉토리
    COPY . .

    # 패키지 다운로드
    RUN npm install --force

    # 빌드 수행
    RUN npm run build

    # 빌드를 위해 nginx를 사용
    FROM nginx:stable-alpine as production-stage

    # 빌드된 폴더를 nginx에서 지정하고 있는 폴더로 이동
    COPY --from=build-stage /var/jenkins_home/workspace/frontend/frontend/build /usr/share/nginx/html

    # 사용할 nginx 설정 파일을 이동
    COPY --from=build-stage /var/jenkins_home/workspace/frontend/frontend/nginx.conf /etc/nginx/conf.d/default.conf

    # 80번 포트를 오픈
    EXPOSE 80

    # 컨테이너 실행시 `nginx -g daemon off`를 수행
    CMD ["nginx", "-g","daemon off;"]
    ```

  - `docker-compose.yml`(Jenkins용)

    ▼ 코드 보기

    ```
    version: "3"

    services:
      jenkins:
        image: jenkins/jenkins:jdk17
        container_name: jenkins_cicd
        volumes:
          - /var/run/docker.sock:/var/run/dodcker.sock
          - /jenkins:/var/jenkins_home
        ports:
          - "9090:8080"
        privileged: true
        user: root
    ```

    - jenkins: 만들어지는 컨테이너의 이름. 자유롭게 설정 가능
    - image: 컨테이너에 올라갈 이미지

- volumes: 컨테이너가 내려갔을 때 파일이 소실되는걸 방지하기 위한 로컬 저장 경로
- ports: {외부 ↔ 컨테이너 통신용 포트}:{컨테이너 ↔ 프로그램 통신용 포트}
- privileged: 컨테이너가 시스템 주요 자원에 접근할 수 있도록 권한을 부여하는 옵션
- user: 컨테이너 접속시 디폴트 사용자
- `docker-compose.yml` 사용 명령어

```
// docker-compose 명령어로 docker-compose.yml 파일을 사용할 수 있음
// 기본적으로 현재 폴더를 기준으로 하고, -f 옵션을 통해 파일의 위치를 지정할 수 있음
// up으로 컨테이너를 올릴 수 있고, down으로 컨테이너를 내릴 수 있음
// --build는 캐싱된 이미지를 사용하지 않고 무조건 빌드를 하고 시작한다는 의미
// -d는 컨테이너를 백그라운드에서 올리겠다는 의미
docker-compose -f docker-compose.yml up --build
docker-compose -f docker-compose.yml up -d
docker-compose -f docker-compose.yml down
```

NGINX

- 설치

```
apt install nginx
```

- NGINX 상태확인/실행/중지

```
// nginx 상태확인
service nginx status

// nginx 실행
service nginx start

// nginx 재실행
service nginx restart

// nginx 중지
service nginx stop
```

nginx 실행 과정에서 80번 포트가 이미 사용중이라고 나오면서 실행에 실패할 경우 다음 명령어를 수행

```
# 80번 포트를 사용중인 프로세스들을 확인
lsof -i :80

# 80번 포트를 사용중인 프로세스들을 종료
fuser -k 80/tcp
```

- 삭제

진행 과정에서 문제가 생겨 nginx를 완전히 삭제할 필요가 있을 경우 다음 명령어 수행

```
apt-get remove --purge nginx nginx-full nginx-common
```

Certbot(https 설정)

- 사전 준비

https가 적용될 서버 블록이 필요하기 때문에 우선 NGINX를 이용해서 서버 블록을 지정

```
// 도메인 이름으로는 j7d108.p.ssafy.io를 사용
mkdir -p /var/www/{도메인이름}/html
chown -R $USER:$USER /var/www/{도메인이름}/html
chmod -R 755 /var/www/{도메인이름}
```

`/etc/nginx/sites-available`에 `j7d108.p.ssafy.io.conf`를 만들고 아래 명령어로 `/etc/nginx/sites-enabled`에도 심볼릭 링크로 연결

```
ln -s /etc/nginx/sites-available/j7d108.p.ssafy.io.conf /etc/nginx/sites-enabled/
```

- 설치

```
snap install --classic certbot
```

- SSL 인증서 발급

```
certbot certonly --nginx -d j7d108.p.ssafy.io
```

명령어를 입력하면 동의 및 설정 절차를 가질 수 있음

성공적으로 설정이 완료되면 `/etc/letsencrypt/live/{도메인이름}` 디렉토리에 다음 파일들이 생김

- `cert.pem`: yourdomain.com의 인증서(public key)
- `chain.pem`: Let's emcrypt의 Intermediate 인증서
- `fullchain.pem`: `cert.pem`과 `chain.pem`을 합친 결과
- `privkey.pem`: `cert.pem` publickey에 대응하는 비밀키

그리고, sites-available에 있는 default파일이나 도메인 이름에 해당하는 파일에 certbot이 자동으로 작성한 코드가 생김

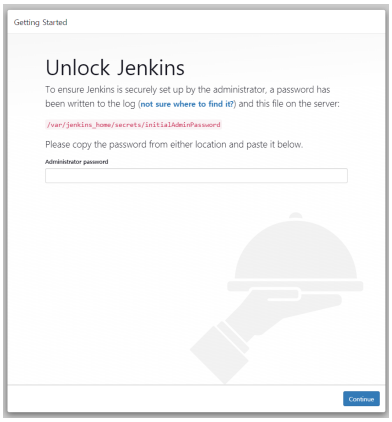
그대로 사용해도 괜찮지만, 만약 문제가 생길 경우 직접 수정해서 사용해도 무방함

ex1) 기본 리다이렉션 설정은 301로 되어있지만, 필요에 따라 308로 변경 가능

ex2) certbot이 작성한 코드를 전부 지우고 원하는 방식으로 코드를 구성

Jenkins

- 여기서 작성한 `docker-compose.yml` 파일을 `docker-compose up -d`로 올리면 젠킨스 컨테이너를 올릴 수 있음
- 컨테이너가 올라가면 브라우저에서 `j7d108.p.ssafy.io:9090`으로 젠킨스 페이지에 접속 가능



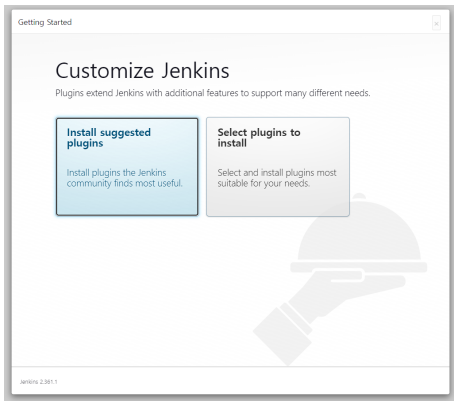
- 젠킨스 페이지에 접속하면 암호를 입력해야하는데 암호는 다음 명령어로 확인 가능

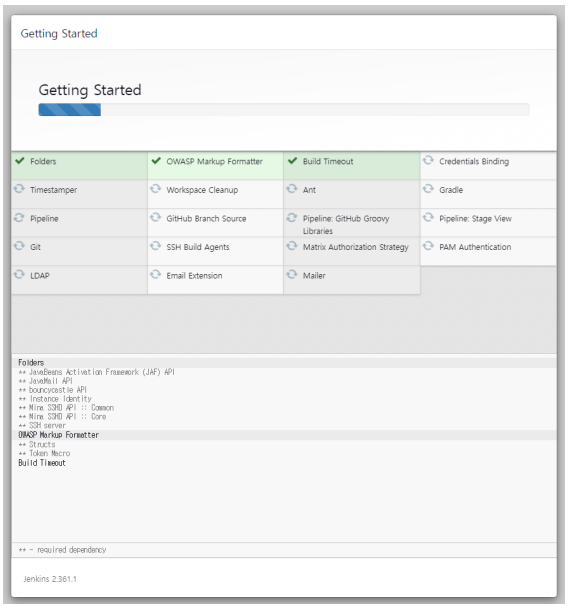
```
docker logs {컨테이너 이름}

# 또는

docker-compose logs
```

```
jenkins_cicd | 2022-10-06 07:55:21.853+0000 [id=28]     INFO    jenkins.install.SetupWizard#init:
jenkins_cicd |
jenkins_cicd | ************************************************************
```

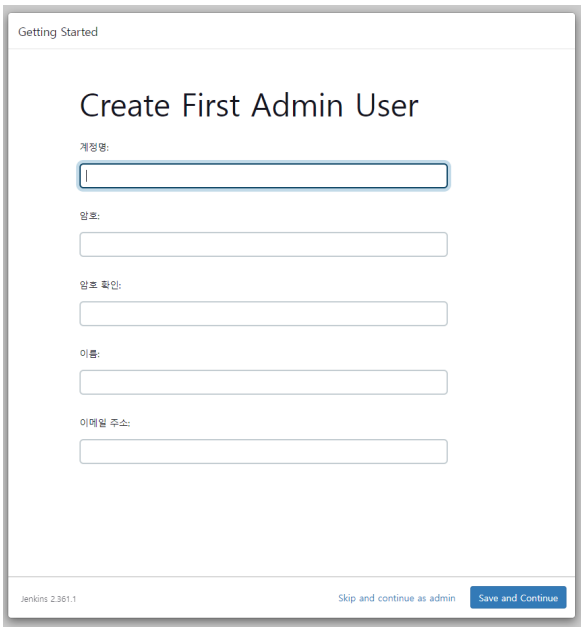
```
jenkins_cicd | *************************************************************
jenkins_cicd | *************************************************************
jenkins_cicd |
jenkins_cicd | Jenkins initial setup is required. An admin user has been created and a password generated.
jenkins_cicd | Please use the following password to proceed to installation:
jenkins_cicd |
jenkins_cicd | {암호}
jenkins_cicd |
jenkins_cicd | This may also be found at: /var/jenkins_home/secrets/initialAdminPassword
jenkins_cicd |
jenkins_cicd | *************************************************************
jenkins_cicd | *************************************************************
jenkins_cicd | *************************************************************
jenkins_cicd |
jenkins_cicd | 2022-10-06 07:55:55.640+0000 [id=29]     INFO    jenkins.InitReactorRunner$1#onAttained: Completed initialization
```

- 암호를 입력하면 플러그인 설치 창이 나오는데, 왼쪽의 `Install suggested plugins`로 플러그인 설치





- 플러그인 설치가 끝나면 계정명, 암호, 이름, 이메일 주소 등을 입력해서 관리자 계정을 설정
  이때 계정명과 암호는 젠킨스 페이지에 로그인 할 때 ID와 비밀번호로 사용됨



- 이후 url은 수정할 필요 없이 `save and finish`를 하면 젠킨스 초기 설정이 끝나고 젠킨스 페이지에 접속 가능
- 젠킨스 메인에서 `Jenkins 관리` → `플러그인 관리` → `설치 가능`으로 이동해서 다음 플러그인들을 설치
  - Gitlab
  - Gitlab API
  - Gitlab Authentication
  - Generic Webhook
  - Docker
  - Docker Commons
  - Docker Pipeline
  - Docker API
  - Publish Over SSH

DataBase

MySQL

- AWS EC2 서버에서 MySQL Docker image 받아서 container 실행

```
sudo docker run --name mysql_test -e MYSQL_ROOT_PASSWORD=password -d -p 3306:3306 mysql
```

- 실행 중인 MySql container 접속

```
sudo docker exec -it mysql_test bash
```

- root 계정 생성 및 password 설정

```
mysql -uroot -p
password
create user 'root'@'localhost' identified by '비밀번호';

mysql -u root -p
show databases;
use mysql;
select user,host from user;
alter user 'root'@'localhost' identified with mysql_native_password by '비밀번호'
flush privileges;
```

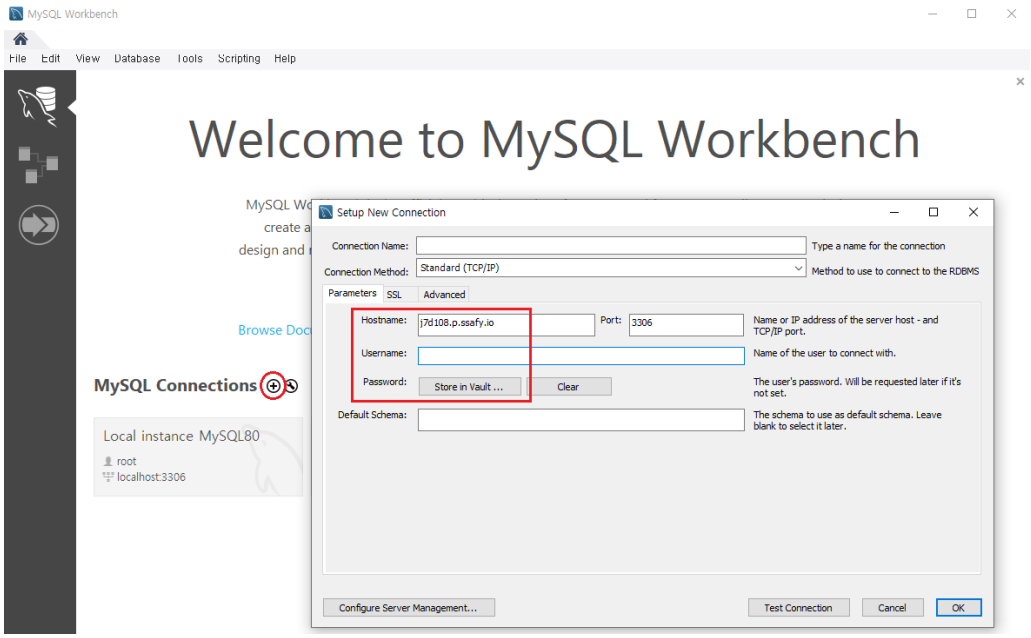
- 사용할 계정 생성

```
create user '계정명'@'아이피주소' identified by '비밀번호';
flush privileges;
```

- 계정에 권한 부여

```
grant all privileges on *.* to '계정명'@'아이피주소';
flush privileges;
```

- MySQL Workbench 접속



Hostname에 배포 도메인 이름, Username, Password에 생성한 계정명, 비밀번호 입력해서 접속

Redis

- 설치

```
# 패키지 업데이트
apt-get update

# Redis 설치
apt-get install redis-server

# Redis 버전 확인
redis-server --version
```

- Redis 상태확인/실행/중지

```
// redis 상태확인
service redis status

// redis 실행
service redis start

// redis 재실행
service redis restart

// redis 중지
service redis stop
```

- 설정 변경

  `/etc/redis/redis.conf` 파일을 수정해서 설정을 변경할 수 있고, 설정을 수정한 뒤에는 redis를 재시작해야 함

  단, 파일 내용이 길어서 수정하고 싶은 내용을 찾기 어려울 수 있으므로 `grep` 명령어를 사용하는 것을 추천

  - 최대 사용 메모리(maxmemory)와 메모리 초과 시 삭제 정책(maxmemory-policy) 설정

    `grep -n "maxmemory" /etc/redis/redis.conf` 명령어로 수정하고 싶은 옵션의 위치를 찾은 뒤 수정

```
root@ip-172-26-3-19:/home/ubuntu# grep -n "maxmemory" /etc/redis/redis.conf
545:# according to the eviction policy selected (see maxmemory-policy).
555:# WARNING: If you have replicas attached to an instance with maxmemory on,
563:# limit for maxmemory so that there is some free RAM on the system for replica
566:# maxmemory <bytes>
568:# MAXMEMORY POLICY: how Redis will select what to remove when maxmemory
597:# maxmemory-policy noeviction
```

```
608:# maxmemory-samples 5
610:# Starting from Redis 5, by default a replica will ignore its maxmemory setting
622:# memory than the one set via maxmemory (there are certain buffers that may
626:# the configured maxmemory setting.
628:# replica-ignore-maxmemory yes
654:# 1) On eviction, because of the maxmemory and maxmemory policy configurations,
1040:#  e     Evicted events (events generated when a key is evicted for maxmemory)
1255:# Redis LFU eviction (see maxmemory setting) can be tuned. However it is a good
```

위의 경우 566번 줄과 597번 줄의 주석을 풀고 내용을 수정하거나 그 아래에 내용을 작성해서 설정을 변경할 수 있음

```
# 메모리를 1g로 설정
maxmemory 1g

# 가장 오래된 데이터를 삭제
maxmemory-policy allkeys-lru
```

삭제 정책의 종류에 관해서는 아래 링크를 참고

> Key eviction
> Overview of Redis key eviction policies (LRU, LFU, etc.) When Redis is used as a cache, it is often convenient to let it automatically evict old data as you add new data. This behavior is well known in the developer community, since it is the default behavior for the popular memcached system.
> https://redis.io/docs/manual/eviction/

- springboot에서 redis를 사용하기 위한 설정
  - build.gradle 의존성 추가

```
implementation 'org.springframework.boot:spring-boot-starter-data-redis'
```

  - application.yml 설정 추가

```
spring:
  cache:
    type: redis
  redis:
    host: j7d108.p.ssafy.io
    port: 6379
```

프로퍼티 정의

nginx 설정 파일

1. `nginx.conf` (nginx 디폴트 설정)

   `/etc/nginx` 폴더에 기본으로 존재하는 디폴트 설정 파일

   설처했을 때 설정에서 변경 X

   ▼ 코드 보기

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
        worker_connections 768;
        # multi_accept on;
}

http {

        ##
        # Basic Settings
        ##

        sendfile on;
```

```
        tcp_nopush on;
        tcp_nodelay on;
        keepalive_timeout 65;
        types_hash_max_size 2048;
        # server_tokens off;

        # server_names_hash_bucket_size 64;
        # server_name_in_redirect off;

        include /etc/nginx/mime.types;
        default_type application/octet-stream;

        ##
        # SSL Settings
        ##

        ssl_protocols TLSv1 TLSv1.1 TLSv1.2 TLSv1.3; # Dropping SSLv3, ref: POODLE
        ssl_prefer_server_ciphers on;

        ##
        # Logging Settings
        ##

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

        ##
        # Gzip Settings
        ##

        gzip on;

        # gzip_vary on;
        # gzip_proxied any;
        # gzip_comp_level 6;
        # gzip_buffers 16 8k;
        # gzip_http_version 1.1;
        # gzip_types text/plain text/css application/json application/javascript text/xml application/xml application/xml+rss

        ##
        # Virtual Host Configs
        ##

        include /etc/nginx/conf.d/*.conf;
        include /etc/nginx/sites-enabled/*;
}


#mail {
#       # See sample authentication script at:
#       # http://wiki.nginx.org/ImapAuthenticateWithApachePhpScript
#
#       # auth_http localhost/auth.php;
#       # pop3_capabilities "TOP" "USER";
#       # imap_capabilities "IMAP4rev1" "UIDPLUS";
#
#       server {
#               listen     localhost:110;
#               protocol   pop3;
#               proxy      on;
#       }
#
#       server {
#               listen     localhost:143;
#               protocol   imap;
#               proxy      on;
#       }
#}
```

2. `j7d108.p.ssafy.io.conf` (서버 설정)

   `/etc/nginx/sites-available` 폴더에 도메인 이름에 맞춰서 새로 만들어 줄 설정 파일

   `/etc/nginx/sites-enabled` 폴더에도 심볼릭 링크로 연결되어 있음

   certbot을 이용해서 https 설정을 할 때 자동으로 작성되는 코드를 사용하지 않고, 직접 작성해서 사용

   ▼ 코드 보기(주석 달아야 함)

```
upstream backend {
        ip_hash;
        server  172.26.3.19:8080;
}
```

```
upstream frontend {
        ip_hash;
        server  172.26.3.19:3000;
}


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

        server_name     j7d108.ssafy.io;

        if ($host = j7d108.p.ssafy.io) {
                return 308 https://$host$request_uri;
        }
}

server {
        listen  443     ssl;
        server_name     j7d108.p.ssafy.io;

        ssl_certificate /etc/letsencrypt/live/j7d108.p.ssafy.io/fullchain.pem;
        ssl_certificate_key     /etc/letsencrypt/live/j7d108.p.ssafy.io/privkey.pem;

        location /api {
                proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;

                proxy_pass      http://backend;

                proxy_redirect  off;

                proxy_set_header        X-Real-IP       $remote_addr;

                proxy_set_header        X-Forwarded-Proto       $scheme;

                proxy_set_header        Host    $http_host;

                add_header      P3P 'CP="ALL DSP COR PSAa PSDa OUR NOR ONL UNI COM NAV"';
        }

        location /resource {
                root    /home/ubuntu/gamulgamul/;
        }

        location / {
                proxy_pass      http://frontend;
        }
}
```

3. `nginx.conf` (프론트엔드 nginx 설정)

   Frontend `Dockerfile` 에서 사용하는 nginx 설정 파일

   반드시 `Dockerfile` 에서 명시하는 경로에 있어야 함

   ▼ 코드 보기

```
server {
    listen 80;

    location / {
        root    /usr/share/nginx/html;
        index   index.html;
        try_files $uri $uri/ /index.html;
    }
}
```

백엔드 설정 파일

아래 파일들 모두 Jenkins 컨테이너 안에 있는 backend 아이템에 넣어야 함

1. `application.yml` (springboot 설정)

   `/src/main/resources` 에 위치하는 springboot 설정 파일

   ▼ 코드 보기

```
spring:
  datasource:
    url: jdbc:mysql://j7d108.p.ssafy.io:3306/gamulgamul_test3?useSSL=false&characterEncoding=UTF-8&useUnicode=true&allowPubli
    username: ssafy
    password: cvgkbhs234r#8tdx
    driver-class-name: com.mysql.cj.jdbc.Driver
    hikari:
      data-source-properties:
        rewriteBatchedStatements: true

  cache:
    type: redis
  redis:
    host: j7d108.p.ssafy.io
    port: 6379

  jpa:
    hibernate:
      ddl-auto: none
    properties:
      hibernate:
        jdbc:
          batch_size: 500
        format_sql: true
        use_sql_comments: true
        default_batch_fetch_size: 500
        order_updates: true
        order_inserts: true
    database: mysql
    database-platform: org.hibernate.dialect.MySQL5InnoDBDialect

  servlet:
    multipart:
      max-file-size: 100MB
      max-request-size: 100MB

  profiles:
    include: API-KEY

server:
  servlet:
    context-path: /api

logging.level:
  org.hibernate.SQL: debug
```

2. `application-API-KEY.properties` (Naver 뉴스 검색 API Key)

   Naver 뉴스 검색 API로 정보를 받아오기 위한 cliendid와 secretkey

   `application.yml` 과 같은 경로에 위치

   ▼ 코드 보기

```
clientId = 8dsQLQwo_ame0XieXqz0
clientSecret = 52VY2squXV
```

3. `JwtProperties.java` (Jwt 관련 상수들이 정의된 파일)

   Jwt에서 사용하는 상수들을 정의하고 사용하기 위한 파일

   `/src/main/java/com/lemonmul/gamulgamul/security/jwt에 위치`

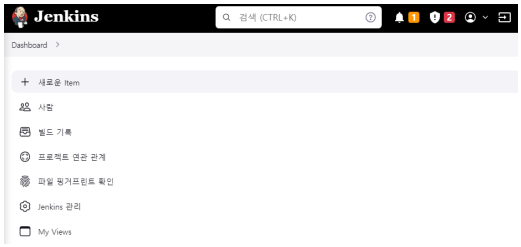
   ▼ 코드 보기

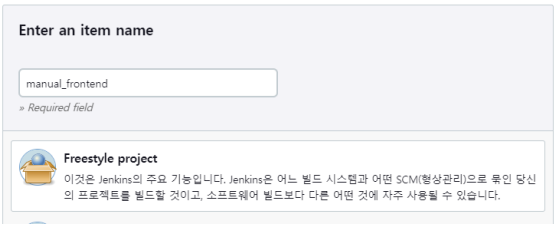
```
package com.lemonmul.gamulgamul.security.jwt;

public interface JwtProperties {
    String SECRET= "&er768uguy^%^*Ii";
    // 60 * 1000L = 1분
    long ACCESS_EXPIRATION_TIME= 1440 * 60 * 1000L;
    long REFRESH_EXPIRATION_TIME= 14 * 1440 * 60 * 1000L; // 1440분 = 24시간
    String TOKEN_PREFIX= "Bearer ";
    String HEADER_STRING= "Authorization";
}
```

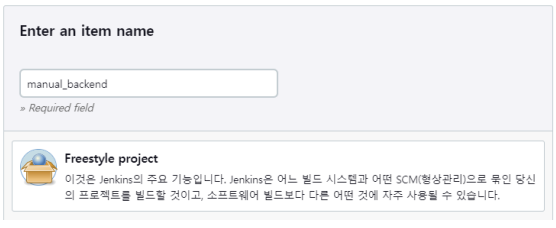
Jenkins 설정

소스 코드 관리

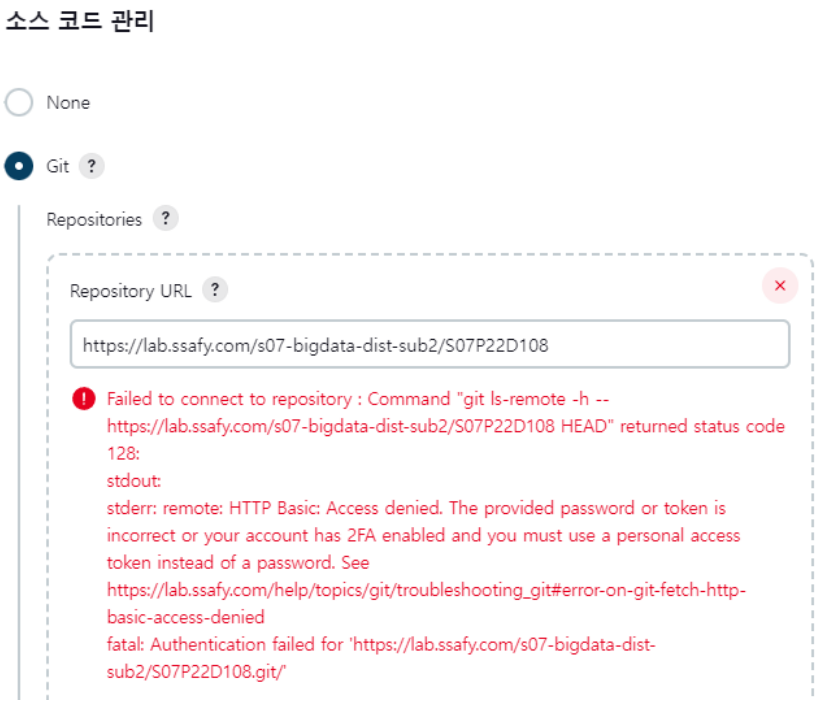
- 젠킨스 메인페이지에서 `새로운 Item`을 누르고 frontend와 backend 빌드를 위한 아이템을 `Freestyle project`로 각각 생성(아이템 이름은 자유롭게 설정)



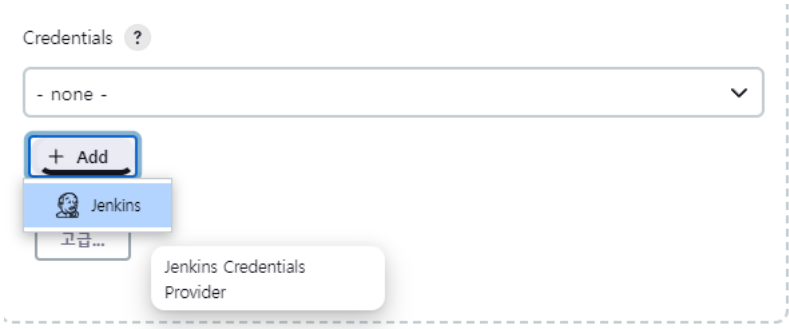




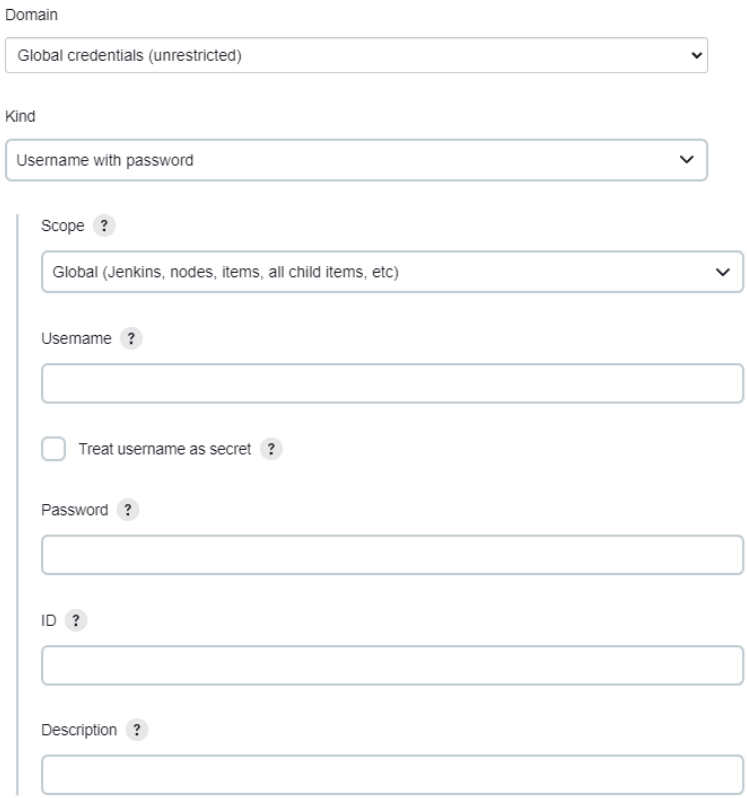
- 소스 코드 관리 탭에서 `Git`을 클릭하고 `Repository URL`에 gitlab 프로젝트 주소를 입력



- `Credentials에서` `Add` → `Jenkins`를 클릭

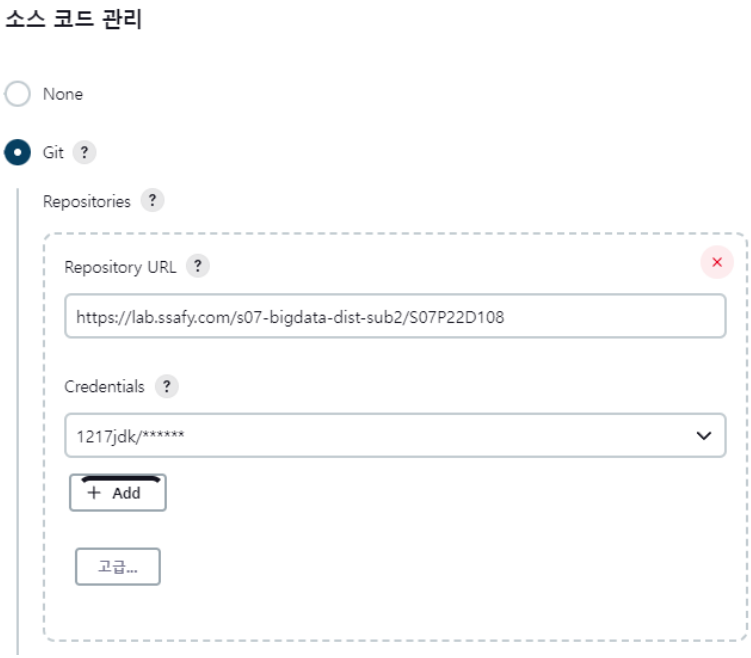


- Credential 관련 설정을 입력

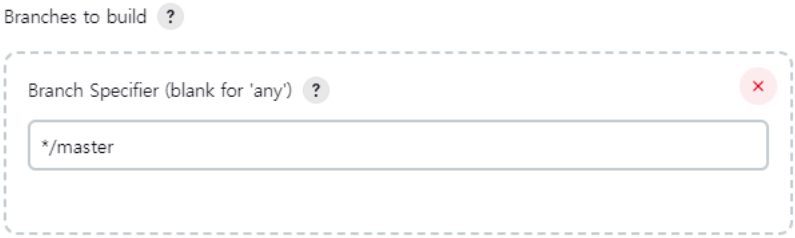


  - `Username` : Gitlab 아이디
  - `Password` : Gitlab 비밀번호
  - `ID` : Credential을 구분하기 위한 이름

⇒ 아래처럼 에러 메시지가 없어지면 성공적으로 Credential 등록에 성공한 것

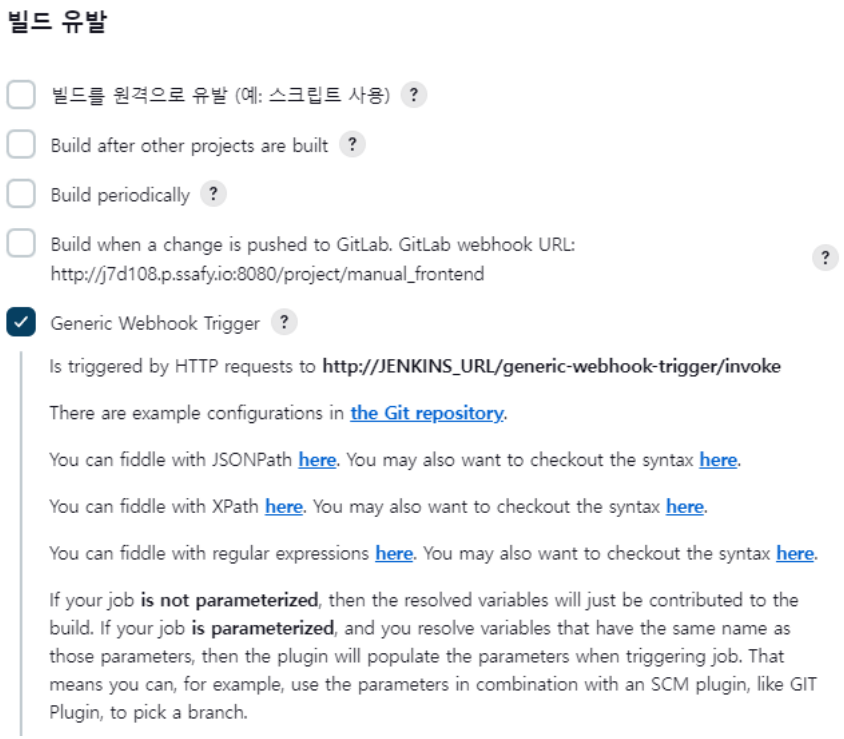


- 빌드에 사용할 Branch를 설정



Webhook

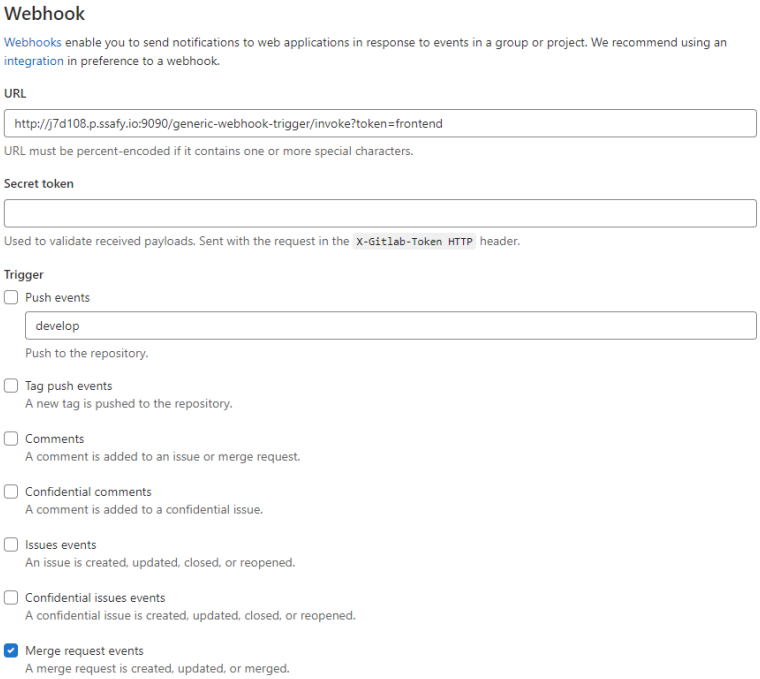
- `빌드 유발`에서 `Generic Webhook Trigger`를 선택



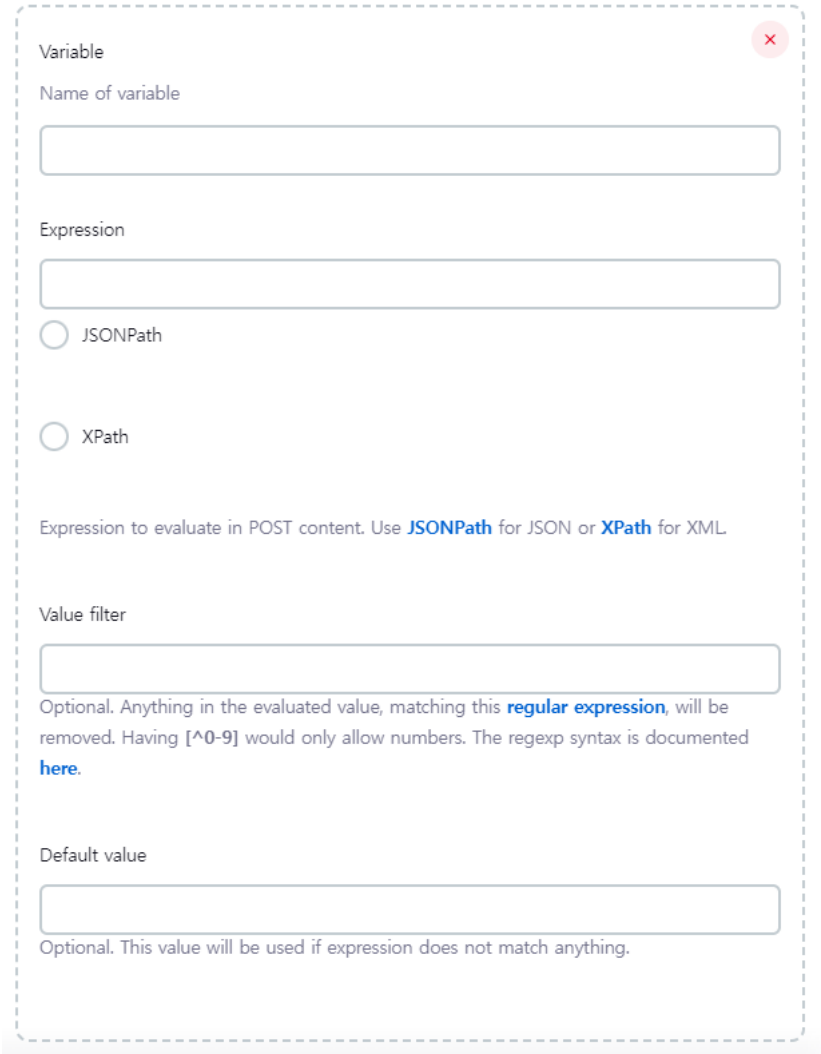
- Webhook의 트리거로 사용할 토큰을 설정
  - manual_frontend는 frontend, manual_backend는 backend로 설정함

- Webhook 트리거 주소를 Gitlab에 등록
  - Gitlab 프로젝트의 설정에서 Webhook으로 이동

  - `URL`에 `http://{젠킨스주소}/generic-webhook-trigger/invoke?token={토큰이름}` 형식으로 경로를 입력하고, `Trigger`에서 `Merge request events`를 선택



- Post content parameters를 추가
  - Post 요청을 받았을 때 요청에 있는 값들 중에서 트리거로 사용할 변수들을 지정하는 것



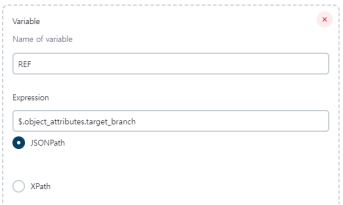
- `Name of variable`: 사용자가 설정하는 변수명
- `Expression`: `JSONPath` 또는 `XPath` 형식으로 나타나는 표현식
  - `JSONPath`: JSON 객체 탐색에 사용되는 형식
  - `XPath`: XML 데이터 탐색에 사용되는 형식

  ⇒ Gitlab의 Webhook으로 받는 정보를 JSON으로 처리할 것이므로 `JSONPath`를 사용
- Gitlab의 `merge request events`로 받는 데이터의 형식은 아래 링크를 참고

> Webhook events | GitLab
> Documentation for GitLab Community Edition, GitLab Enterprise Edition, Omnibus GitLab, and GitLab Runner.
> https://docs.gitlab.com/ee/user/project/integrations/webhook_events.html

- Gitlab에서 머지가 완료됐을 때, Label을 기준으로 Frontend와 Backend의 빌드 분기가 발생하도록 설정하기 위해 다음 3가지 parameter를 받아서 사용

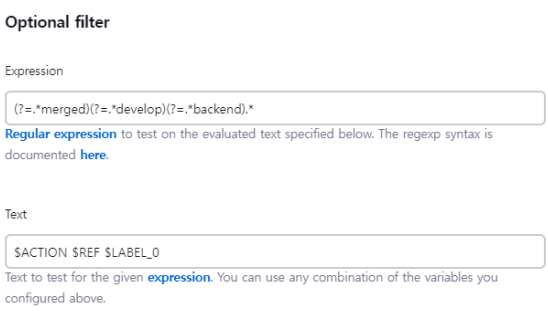
Label



타겟 branch

머지 결과

- Optional filter를 설정해서 parameter의 값에 따라 트리거 발생 여부를 결정



  - Expression: 정규식으로 정의된 트리거 발생 조건
  - Text: 표현식의 각 부분에 매핑될 사용자 정의 변수들

빌드

- 젠킨스에서 도커 빌드를 하기 위해 젠킨스 컨테이너에 도커를 설치
  - EC2에서 젠킨스 컨테이너에 접속

```
# 도커 컨테이너에 접속하는 명령어
docker exec -it {컨테이너명} bash

# jenkins 컨테이너 명이 jenkins_cicd일 경우
docker exec -it jenkins_cicd bash
```

  - 젠킨스 컨테이너 안에서 도커 설치

    여기와 동일한 과정으로 도커 설치

- 젠킨스 아이템 구성 페이지로 돌아가서, `Build Steps`에서 `Execute shell`을 선택하고 build에 사용할 명령어를 입력

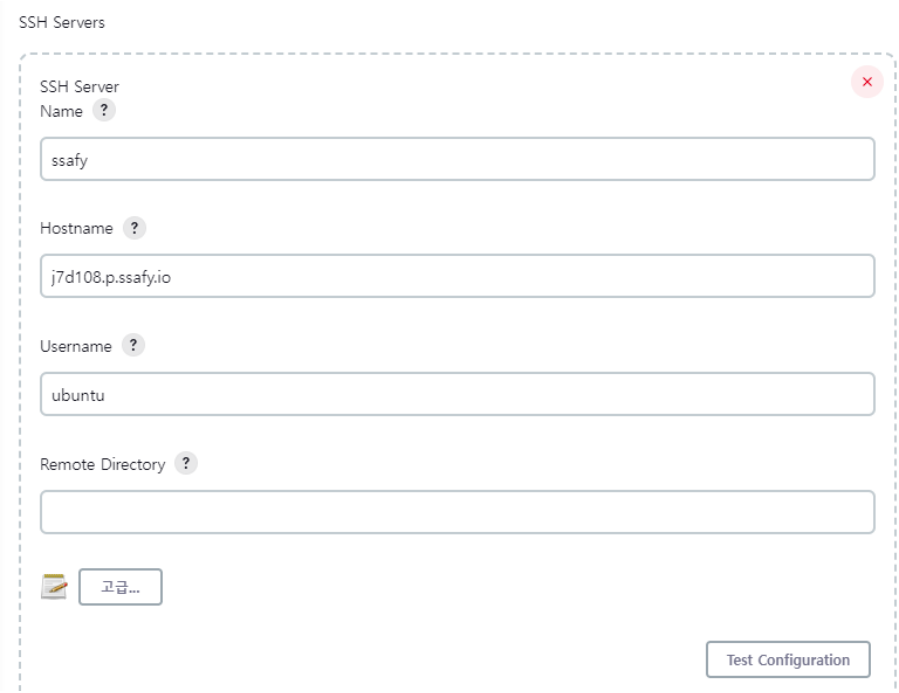
```
# Frontend Build Steps

# 사용하지 않는 이미지 강제 삭제
docker image prune -a --force

# 빌드가 완료된 이미지들을 저장할 폴더를 생성
mkdir -p /var/jenkins_home/images_tar

# 빌드를 수행할 폴더로 이동
cd /var/jenkins_home/workspace/frontend/frontend

# -t 옵션으로 태그명을 설정해서 현재 폴더에 대한 이미지 빌드 수행
docker build -t react .

# 도커 이미지를 지정한 경로에 tar 형식으로 저장
docker save react > /var/jenkins_home/images_tar/react.tar
```

```
# Backend Build Steps

# 사용하지 않는 이미지 강제 삭제
docker image prune -a --force

# 빌드가 완료된 이미지들을 저장할 폴더를 생성
mkdir -p /var/jenkins_home/images_tar

# 빌드를 수행할 폴더로 이동
cd /var/jenkins_home/workspace/backend/backend

# -t 옵션으로 태그명을 설정해서 현재 폴더에 대한 이미지 빌드 수행
docker build -t spring .
```

```
# 도커 이미지를 지정한 경로에 tar 형식으로 저장
docker save spring > /var/jenkins_home/images_tar/spring.tar
```

배포

- 젠킨스에서 SSH 명령어 전송을 하기 위해 SSH 연결 설정
  - `Jenkins 관리` → `시스템 설정`에서 Publish over SSH 설정

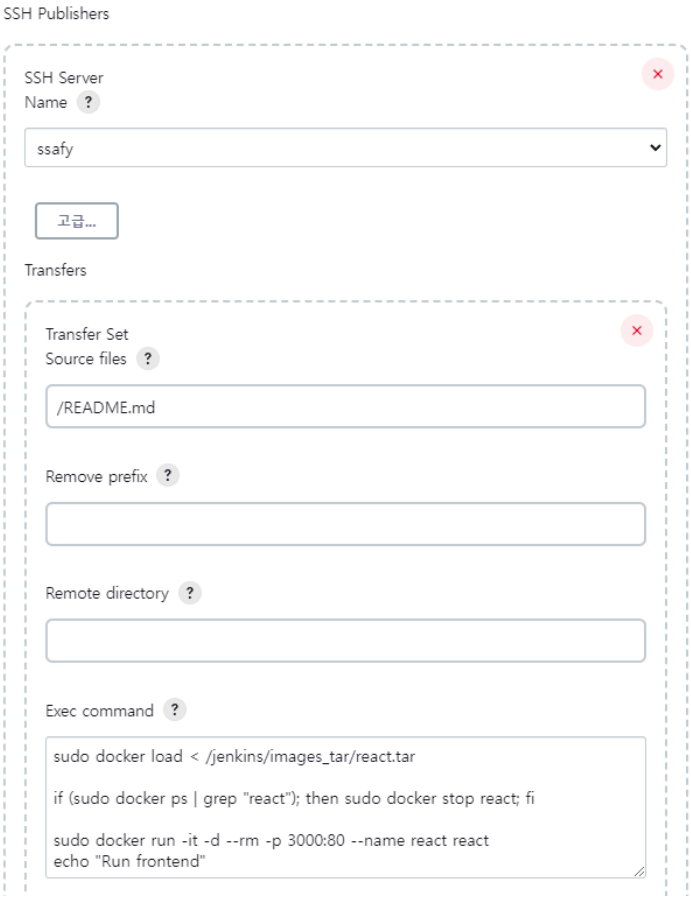


    - Name: SSH 서버를 식별하기 위한 사용자 지정 이름(원하는 이름으로 정하면 됨)
    - Hostname: EC2 서버 ip 또는 도메인
    - Username: EC2 접속 계정명
  - 고급 버튼을 눌러서 나온 `Use password authentication, or use different key`를 체크하고 key에 서버 pem키를 등록

    `cat` 명령어, vscode, 메모장 등으로 pem 키의 내용을 읽고 붙여넣으면 됨

    반드시 BEGIN~라인과 END~라인을 포함해야 함

    ```
    -----BEGIN RSA PRIVATE KEY-----
    MIIEpAIBAAKCAQEAiovhcapKL0dcTbpdqtNOYz4d5iOe2Admx1HUejFSfC9M2ObY
    Euiur+igwGI/sRIQu1RhibZn/2Uj7klWXFxlVItU1lnuKh7TEA4MZ6XOI3bbFv/a
    - 중략 -
    WcuLoiPG6VOe+fuPUuN8fhqo0QoGJUeiYHnvYV2+GPdv+fxSexMhjO2c1jHkcwI6
    5s+8s2DVhYaSaITMWcXjLXgZKtvl0cIp4F/woxf4HAHVDsl+iMTy7g==
    -----END RSA PRIVATE KEY-----
    ```
  - `Test Configuration` 버튼을 클릭했을 때 Success가 나오면 성공적으로 연결된 것
- 젠킨스 아이템 구성 페이지로 돌아가서, 빌드 후 조치에 `Send build artifacts over SSH`를 추가



- SSH Server의 `Name`을 방금 설정한 서버로 선택
- `Source files`는 사용할 속성은 아니지만 필수 입력이므로 아무 내용이나 입력하면 됨
- `Exec command`는 젠킨스에서 SSH 명령어를 전송하는 부분이므로 다음 명령어를 입력

```
# Frontend Exec command

# docker 이미지를 읽어오는 부분
sudo docker load < /jenkins/images_tar/react.tar

# 현재 올라가 있는 도커 컨테이너 중에 react라는 이름의 컨테이너가 있으면 해당 컨테이너를 종료시킴
if (sudo docker ps | grep "react"); then sudo docker stop react; fi

# react 이미지를 react라는 이름의 컨테이너로 올림
# -it: 사용자가 입출력 할 수 있는 가상 터미널로 올라가기 때문에 컨테이너 쉘과 직접 상호작용 가능해짐
# -d: 컨테이너를 백그라운드로 올림
# --rm: 컨테이너를 종료하면 자동으로 삭제됨
# -p 컨테이너에서 사용할 포트를 지정
# --name: 컨테이너의 이름을 설정
sudo docker run -it -d --rm -p 3000:80 --name react react
```

```
# Backend Exec command

# docker 이미지를 읽어오는 부분
sudo docker load < /jenkins/images_tar/spring.tar

# 현재 올라가 있는 도커 컨테이너 중에 react라는 이름의 컨테이너가 있으면 해당 컨테이너를 종료시킴
if (sudo docker ps | grep "spring"); then sudo docker stop spring; fi

# react 이미지를 react라는 이름의 컨테이너로 올림
# -it: 사용자가 입출력 할 수 있는 가상 터미널로 올라가기 때문에 컨테이너 쉘과 직접 상호작용 가능해짐
# -d: 컨테이너를 백그라운드로 올림
# --rm: 컨테이너를 종료하면 자동으로 삭제됨
# -p 컨테이너에서 사용할 포트를 지정
# --name: 컨테이너의 이름을 설정
sudo docker run -it -d --rm -p 8080:8080 --name spring spring
```

빌드&배포 관련 모든 설정이 끝났기 때문에 Gitlab에서 frontend 또는 backend 라벨을 붙여서 머지를 성공하면 해당 라벨에 맞는 아이템이 빌드 및 배포됨

제플린 & Spark 설정

서버주소 & 환경변수 요약

(1) 서버주소

- Private
  - 환경설정에서 사용

- Public
  - 웹에서 접속시 이용

(2) 환경변수 설정 요약

- `$ vim /etc/environment`

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
:/usr/lib/jvm/java-8-openjdk-amd64/bin
:/usr/local/hadoop/bin
:/usr/local/hadoop/sbin
:/usr/local/spark/bin
:/usr/local/spark/sbin
:/usr/bin/python3
:/zeppelin-0.10.1-bin-all/bin"

JAVA_HOME="/usr/lib/jvm/java-8-openjdk-amd64"
# HADOOP_HOME="/usr/local/hadoop"
SPARK_HOME="/usr/local/spark"
# ZOOKEEPER_HOME="/usr/local/zookeeper"
```

[과정 1] 서버접속 후 업데이트

- EC2 서버접속

```
# PEM 파일 폴더에서
ssh -i J7D108T.pem ubuntu@j7d108.p.ssafy.io
```

- root로 들어가기

```
su root
password : cvgkbhs234r#8tdx
```

- 업데이트

```
# EC2 Ubuntu terminal

# 업데이트 목록 갱신
sudo apt-get -y update
# 현재 패키지 업그레이드
sudo apt-get -y upgrade
# 신규 업데이트 설치
sudo apt-get -y dist-upgrade
```

```
# 필요 라이브러리 설치
sudo apt-get install -y vim wget unzip ssh openssh-* net-tools
```

[과정 2] Java 설치

- Java 설치

```
# EC2 Ubuntu terminal

# Java 8 설치
sudo apt-get install -y openjdk-8-jdk

# Java 버전 확인
java -version
```

- 경로 확인 for path

```
# Java 경로 확인
sudo find / -name java-8-openjdk-amd64 2>/dev/null

>>>

/usr/share/gdb/auto-load/usr/lib/jvm/java-8-openjdk-amd64
/usr/lib/jvm/java-8-openjdk-amd64
```

[과정 3] Spark 설치 및 실행

- 설치

```
# Spark 3.2.2 설치
sudo wget https://dlcdn.apache.org/spark/spark-3.2.2/spark-3.2.2-bin-hadoop3.2.tgz
# Spark 3.2.2 압축 해제
sudo tar -xzvf spark-3.2.2-bin-hadoop3.2.tgz -C /usr/local
# Spark 디렉토리 이름 변경
sudo mv /usr/local/spark-3.2.2-bin-hadoop3.2 /usr/local/spark
```

- 마스터 실행

```
start-master.sh
# 종료는 stop-master.sh
sudo ss -tunelp | grep 8080
```

- 워커실행

```
start-worker.sh spark://hadoop-virtual-machine:7077
# 종료는 stop-worker.sh
```

- web ui 접속
  - `http://3.36.106.26:18080/` 접속

- mlocate 설치

▼ 활용

find 명령어는 하나하나 검사하기 때문에 속도가 느림

단순한 파일 검색이 필요한 경우 locate 사용

만들어진 db에서 찾기 때문에 빠름 하지만 파일이 많이 삭제되거나 이동되었을 때

locate를 사용한다면 적절한 위치를 찾기 어려울 수 있음 (locate 전에 updatedb 실행해줌)

```
sudo apt -y install mlocate
sudo updatedb
locate start-worker.sh
```

- spark shell 실행

```
# pyspark 실행
pyspark

# scala 실행
spark-shell
```

[과정 4] 제플린 설치 및 실행

- https://zeppelin.apache.org/download.html 에서 zeppelin-버전-bin-all-tgz 다운로드 클릭하면 아래의 https 주소 나옴

```
# 설치
sudo wget https://dlcdn.apache.org/zeppelin/zeppelin-0.10.1/zeppelin-0.10.1-bin-all.tgz

# 압축해제
tar -vxzf zeppelin-0.10.1-bin-all.tgz
```

- 해당 경로로 이동 후, 파일 복사

```
cd zeppelin-0.10.1-bin-all/conf
cp zeppelin-site.xml.template zeppelin-site.xml
cp zeppelin-env.sh.template zeppelin-env.sh
```

- 환경변수
  - 추후 Zeppelin 실행 후, 노트북에서 경로설정 해주는 부분과 일치해야 함!

```
vim zeppelin-env.sh
export SPARK_HOME=/usr/local/spark
export PYTHONPATH=/usr/local/spark/python
export PYSPARK_PYTHON=/usr/local/spark/python



# Zeppeling notebook 부분 (위와 아래가 일치해야 함)
%spark.conf

SPARK_HOME  /usr/local/spark
PYSPARK_PYTHON /usr/bin/python3
spark.pyspark.python  /usr/bin/python3
```

- 서버주소 및 포트 설정

```
/zeppelin-0.10.1-bin-all/conf$ vim zeppelin-site.xml

# 아래 2 곳만 수정
```

```
<property>
  <name>zeppelin.server.addr</name>
  <value>172.26.3.19</value>
  <description>Server binding address</description>
</property>

<property>
  <name>zeppelin.server.port</name>
  <value>8081</value>
  <description>Server port.</description>
</property>

...
```

- 실행

```
/zeppelin-0.10.1-bin-all/bin/zeppelin-daemon.sh start

# 중지
/zeppelin-0.10.1-bin-all/bin/zeppelin-daemon.sh stop
```

- 접속

http://3.36.106.26:8081/

[과정 5] 제플린과 MySQL 연동

(1) Interpreter 생성 공식 문서

- 공식문서참고
  - 이미 존재하는 MySQL과 연동시, 따로 다운받을 것은 없음
  - 문서 예시

Properties

Name	Value
default.driver	com.mysql.jdbc.Driver
default.url	jdbc:mysql://localhost:3306/
default.user	mysql_user
default.password	mysql_password

Mysql JDBC Driver Docs

Dependencies

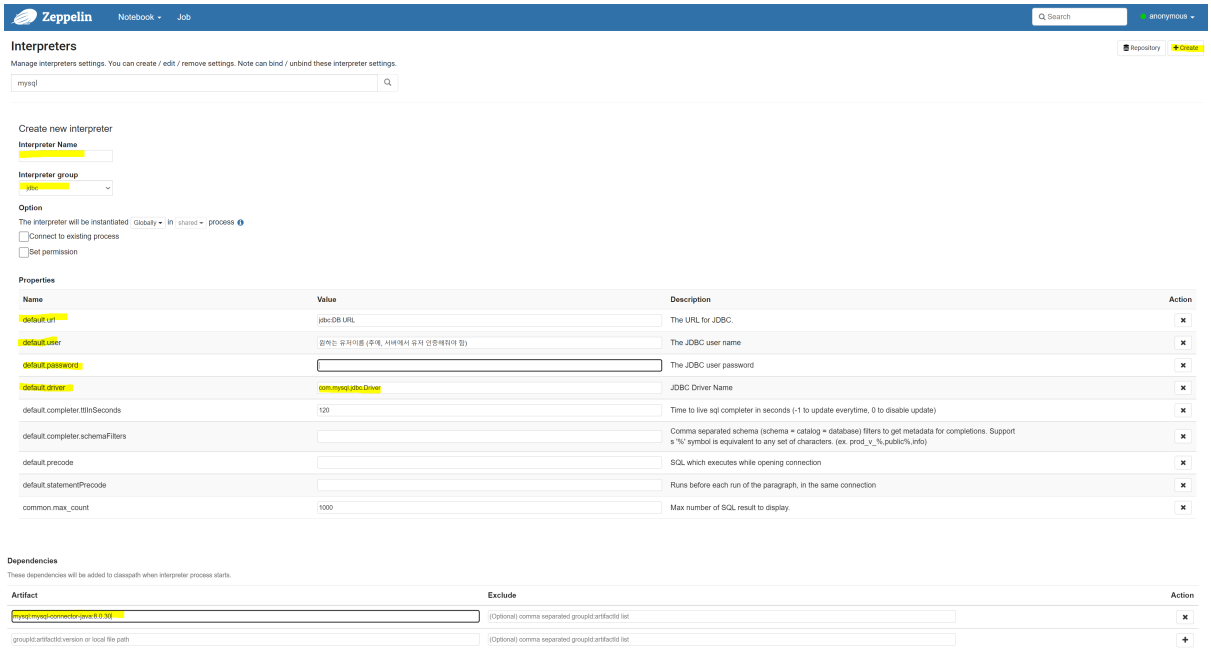
Artifact	Excludes
mysql:mysql-connector-java:5.1.38	

Maven Repository: mysql:mysql-connector-java

-

(2) Interpreter 생성 과정

- 제플린에서 create 클릭 후, interpreter 생성



- 노란부분 입력하면 됨
  - Interpreter Name
    - 스파크 노트북에서 `%InterpreterName` 으로 cell 맨위에서 사용하게 되는 듯
  - default.url
    - DB의 url 입력
  - default.user
    - user name 원하는대로 입력
      - DB에서 사용자 등록 해야 함.
  - default.password
    - DB에서 설정해놓은 비밀번호와 같아야 함
  - default.driver
    - com.mysql.jdbc.Driver 로 설정
  - Dependuencies
    - 원하는 버전 넣는건데, DB의 버전과 같게 넣었음
      - mysql:mysql-connector-java:8.0.30
    - 버전 문서 참고

(3) 제플린에서 MySQL 실행

- **%MySQL** 을 이용해서 MySQL Interpreter 사용가능
- DB에 데이터 넣는 경우, 부모 테이블부터 넣어야 함(참조받는 테이블)
  - category → product → goods 순으로 table 생성해야 함

1. category

```
%MySQL
Insert into category(category_id, img, name)
values(0,'과자&빙과_img', '과자&빙과');
```

2. product

```
%MySQL
Insert into product(product_id, img, measure, name, unit, weight, category_id)
values( 0, '감자칩_img', 'g', '감자칩', 10, 11.333, 0)
```

[과정 6] 제플린에서 pyspark로 MySQL 접근(Read/Write)

(1) 새로운 Table 생성해서, 넣기

```
%pyspark
# 4 - 1 : MySQL에 데이터 넣기 : table 생성됨
category_sdf.select("category_id","img","name").write.format("jdbc")\
    .option("url", "jdbc:mysql://j7d108.p.ssafy.io:3306/spark_test?useSSL=false&characterEncoding=UTF-8&useUnicode=true&allowPublicKey
    .option("driver", "com.mysql.cj.jdbc.Driver")\
    .option("dbtable", "category2") \
    .option("user", "ji")\
    .option("password", "cvgkbhs234r#8tdx").save()
```

(2) 기존의 Table에 넣기

- 참고문서 , https://spark.apache.org/docs/latest/sql-data-sources-jdbc.html

```
# 4 - 2 : MySQL에 데이터 넣기 : 성공 (https://ko.n4zc.com/article/sa4p4m6e.html)
prop = {"user": "ji", "password": "cvgkbhs234r#8tdx", "driver": "com.mysql.cj.jdbc.Driver"}
url = "jdbc:mysql://j7d108.p.ssafy.io:3306/spark_test?useSSL=false&characterEncoding=UTF-8&useUnicode=true&allowPublicKeyRetrieval=tru

category_sdf = spark.read.csv('/DB_data/category_table.csv', header=True, inferSchema=True)
category_sdf.select("category_id","img","name").write.jdbc(\
    url= url,\
    table = "category",\
    mode="append",\
    properties=prop)
```

Frontend

1. `npm install --froce` 을 이용해 `package.json`의 패키지들을 설치한다.

▼ package.json

```
{
  "name": "frontend",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@craco/craco": "^6.4.5",
    "@fortawesome/fontawesome-svg-core": "^6.2.0",
    "@fortawesome/free-solid-svg-icons": "^6.2.0",
    "@fortawesome/react-fontawesome": "^0.2.0",
    "@testing-library/jest-dom": "^5.16.5",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "@types/jest": "^27.5.2",
    "@types/node": "^16.11.57",
    "@types/react": "^18.0.18",
    "@types/react-dom": "^18.0.6",
    "@types/redux-persist": "^4.3.1",
    "flowbite": "^1.5.3",
    "flowbite-react": "^0.1.11",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-loader-spinner": "^5.3.4",
    "react-scripts": "5.0.1",
    "react-slick": "^0.29.0",
    "react-toastify": "^9.0.8",
    "redux-persist": "^6.0.0",
    "slick-carousel": "^1.8.1",
    "typescript": "^4.8.2",
```

```
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "npm run watch:css && craco start",
    "build": "npm run build:css && craco build",
    "test": "craco test",
    "eject": "react-scripts eject",
    "build:css": "postcss src/index.css",
    "watch:css": "postcss src/index.css"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  },
  "devDependencies": {
    "@reduxjs/toolkit": "^1.8.5",
    "@types/axios": "^0.14.0",
    "@types/react-slick": "^0.23.10",
    "@typescript-eslint/eslint-plugin": "^5.36.1",
    "@typescript-eslint/parser": "^5.36.1",
    "autoprefixer": "^10.4.10",
    "axios": "^0.27.2",
    "chart.js": "^3.9.1",
    "craco-alias": "^3.0.1",
    "eslint": "^8.23.0",
    "eslint-config-prettier": "^8.5.0",
    "eslint-plugin-prettier": "^4.2.1",
    "eslint-plugin-react": "^7.31.6",
    "postcss": "^8.4.16",
    "postcss-cli": "^10.0.0",
    "prettier": "^2.7.1",
    "prettier-plugin-tailwindcss": "^0.1.13",
    "react-chartjs-2": "^4.3.1",
    "react-redux": "^8.0.2",
    "react-router-dom": "^6.3.0",
    "redux": "^4.2.0",
    "sass": "^1.54.9",
    "tailwindcss": "^3.1.8",
    "v6": "^0.0.0"
  }
}
```

```
//install package in package.json
// because of older defancay peer, must use --force option
npm install --force
```

2. npm run build 명령어를 통해 `build` 폴더, 파일 생성 `frontend` 폴더 하단에 생성된다.

```
// build frontend source
npm run build
```