

# 공통\_PJT\_대전2반\_B102\_포팅\_매뉴얼

A102 : 주류주아(Jourgeois) - 칵테일 SNS 서비스

삼성청년SW아카데미 대전캠퍼스 7기

공통프로젝트 2022.07.11 ~ 2022.08.19 (6주)

담당 컨설턴트 - 김신일

박승훈(팀장), 고승효, 김낙현, 박재경, 송선아, 전승준

## <목차>

A102 : 주류주아(Jourgeois) - 칵테일 SNS 서비스

삼성청년SW아카데미 대전캠퍼스 7기

공통프로젝트 2022.07.11 ~ 2022.08.19 (6주)

담당 컨설턴트 - 김신일

박승훈(팀장), 고승효, 김낙현, 박재경, 송선아, 전승준

## <목차>

Part 1. 포팅 매뉴얼

1. 프로젝트 기술 스택

1.1 이슈관리 : Jira

1.2 형상관리 : Gitlab

1.3 커뮤니케이션 : Mattermost, Notion, Discord

1.4 개발 환경

1.5 UX/UI

1.6 Database

1.7 Server : AWS EC2 – Ubuntu 20.04 LTS

1.8 상세 내용

2. Property 정의

2.1 Frontend

2.2 Backend

3. 빌드 상세내용

3.1 docker-compose.yml

3.2 Backend: Dockerfile( ./dockerfiles/backend.Dockerfile )

3.3 Frontend: Dockerfile( ./dockerfiles/frontend.Dockerfile )

3.4 frontend - nginx 설정

3.5 Reverse Proxy - nginx 설정

3.6 AWS S3 Bucket – config, access, policy

4. 배포 상세 내용

<수동 배포>

<자동 배포>

5. Ignore 파일

## Part 1. 포팅 매뉴얼

### 1. 프로젝트 기술 스택

**1.1 이슈관리 : Jira**

**1.2 형상관리 : Gitlab**

**1.3 커뮤니케이션 : Mattermost, Notion, Discord**

**1.4 개발 환경**

- OS : Window 10

- IDE
  1. IntelliJ 2022.1.3
  2. Vscode 1.70.0

## 1.5 UX/UI

- figma

## 1.6 Database

- MySQL Workbench

## 1.7 Server : AWS EC2 – Ubuntu 20.04 LTS

- Reverse Proxy : nginx 1.23.1
- WAS : Tomcat Embed Core 9.0.64
- WEB : nginx 1.23.1
- DB : Mariadb 10.8.3
- Redis : redis 7.0.4
- 이미지 스토리지 : AWS S3

## 1.8 상세 내용

### 1.8.1 Backend

- 빌드 : Gradle 7.4.1
- Java : OpenJDK 11
- Spring Boot 2.7.1
- Spring Data JPA 2.7.1
- Lombok 1.18.24

### 1.8.2 Frontend

- Vue : 3.2.37
- Vue-router : 4.1.2
- Vuex : 4.0.2
- Sass : 1.53.0
- Node-sass : 7.0.1
- Ssass-loader: 7.1.0
- Typescript : 4.6.4
- Vite : 3.0.0
- Vue-tsc : 0.38.4

### 1.9 외부 서비스

- Firestore

## 2. Property 정의

### 2.1 Frontend

- .env 파일 (frontend 폴더에 위치시킨다.)

```
// 알림
VITE_FIREBASE_APIKEY = "API 키",

VITE_FIREBASE_AUTHDOMAIN = "",

VITE_FIREBASE_PROJECTID = "",

VITE_FIREBASE_STORAGEBUCKET = "",

VITE_FIREBASE_MESSAGING_SENDERID = "",

VITE_FIREBASE_APPID = "",

VITE_FIREBASE_MEASUREMENTID = ""
```

\* firebase 앱을 Console에서 만들고 내 앱 정보에서 연결에 필요한 정보 확인

## 2.2 Backend

- application.yaml

```
spring:
  mvc:
    static-path-pattern: /img/**
  servlet:
    multipart:
      max-request-size: 10MB
      max-file-size: 10MB
  datasource:
    driver-class-name: org.mariadb.jdbc.Driver

  hikari:
    auto-commit: false
    connection-test-query: SELECT 1
    minimum-idle: 10
    maximum-pool-size: 50
    transaction-isolation: TRANSACTION_READ_UNCOMMITTED
    pool-name: pool-backend
  jpa:
    hibernate:
      ddl-auto: none
    properties:
      hibernate:
        format_sql: true
        show_sql: true
  redis:
    host: redis
    port: 6379

  jwt:
    secret-key: "비밀키" # 키 값을 설정 해야함
    token-validity-in-sec: 30
    refresh-token-validity-in-sec: 604800
  server:
    port: 8080

logging:
  level:
    com:
      amazonaws:
        util:
          EC2MetadataUtils: error
```

- aws.properties

```
cloud.aws.s3.bucket="bucket 이름"
cloud.aws.region.static="bucket 서버 지역"
cloud.aws.credentials.access-key="bucket access-key"
cloud.aws.credentials.secret-key="bucket secret-key"
cloud.aws.s3.bucket.path="bucket 경로"
```

- email.properties

```
mail.smtp.auth=true
mail.smtp.starttls.required=true
mail.smtp.starttls.enable=true
mail.smtp.socketFactory.class=javax.net.ssl.SSLSocketFactory
mail.smtp.socketFactory.fallback=false
mail.smtp.port=465
mail.smtp.socketFactory.port=465

AdminMail.id="이메일 주소"
AdminMail.password="계정 비밀번호"
```

- ServiceKey.json - (주의. 과금 가능성)
1. Firebase 콘솔 에서 프로젝트 생성 (GCP에 기존 프로젝트가 있다면 생략 가능)
  2. 설정 → 프로젝트 설정 → 서비스 계정 → 새 비공개 키 생성
- 위의 설정 파일을 `\src\main\resources`, 또는 `classpath`에 위치 에 위치 시킨다.

## 3. 빌드 상세내용

### 3.1 docker-compose.yml

```
# `Host:` 는 Host Machine 상의 경로를 의미함
version: "3"
services:
  nginxproxy:
    image: nginx:latest
    depends_on:
      - nginx
      - springboot
      - db
      - redis
    ports:
      - "80:80"
      - "443:443"
    restart: always
    volumes:
      # Certbot도 반드시 같은 경로를 마운트 하고 있어야함
      - "Host:인증서 검증을 위해 certbot과 공유하는 경로":"/usr/share/nginx/html"
      - "Host:인증서 경로":"/etc/letsencrypt"
      - "Host:reverse proxy(nginx) 설정 파일 경로":"/etc/nginx/nginx.conf"
  nginx:
    build: ./frontend
    restart: always

  db:
    image: mariadb:latest
    restart: always
    environment:
      MARIADB_DATABASE: "생성할 DB 명"
      MARIADB_USER: "생성할 유저 명"
      MARIADB_PASSWORD: "생성할 유저 비밀번호"
      MARIADB_ROOT_PASSWORD: "root 비밀번호"
    volumes:
      - "Host:생성된 DB meta 정보를 저장할 경로":"/var/lib"
      - "Host:DB 설정 파일 경로":"/etc/mysql/conf.d"
    ports:
      - "3306:3306"

  redis:
    image: redis:alpine #기본 ports: - "6379:6379"

  springboot:
    build: ./backend
    restart: always
    environment:
      SPRING_DATASOURCE_URL: jdbc:mariadb://db:3306/"db명"?useUnicode=true
      SPRING_DATASOURCE_USERNAME: "유저 명"
      SPRING_DATASOURCE_PASSWORD: "유저 비밀번호"
    ports:
      - "8080"
    depends_on:
      - db
      - redis

  certbot:
    depends_on:
      - nginxproxy
    image: certbot/certbot
    volumes:
```

```

- "Host:인증서 위치":/etc/letsencrypt
- "Host:인증서 검증을 위해 certbot과 공유하는 경로":/usr/share/nginx/html
command: certonly --webroot --webroot-path=/usr/share/nginx/html --email "인증서 재발급시 필요한 인증 메일 주소" --agree-tos --no-eff-email

```

### 3.2 Backend: Dockerfile( ./dockerfiles/backend.Dockerfile )

```

FROM openjdk:11-jdk

WORKDIR /app

COPY ./build/*SNAPSHOT.jar application.jar

EXPOSE 8080

CMD ["java", "-jar", "-Duser.timezone=Asia/Seoul", "application.jar"]

```

### 3.3 Frontend: Dockerfile( ./dockerfiles/frontend.Dockerfile )

```

FROM node:lts-alpine as builder

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

RUN npm run build

FROM nginx:stable-alpine as production-build

RUN rm -rf /etc/nginx/conf.d/*

COPY --from=builder /app/nginx /etc/nginx/conf.d

RUN rm -rf /usr/share/nginx/html/*

COPY --from=builder /app/dist /usr/share/nginx/html

EXPOSE 80

ENTRYPOINT ["nginx", "-g", "daemon off;"]

```

### 3.4 frontend - nginx 설정

```

server {

    listen      80;

    server_name localhost;

    location / {

        root    /usr/share/nginx/html;

        index   index.html index.htm;

        try_files $uri $uri/ /index.html;

    }

    error_page   500 502 503 504  /50x.html;

    location = /50x.html {

        root    /usr/share/nginx/html;

    }

}

```

### 3.5 Reverse Proxy - nginx 설정

#### 3.5.1 default.conf ( ./nginx/default\_ssl.conf ) - SSL version

```

user nginx;

worker_processes auto;

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

pid /var/run/nginx.pid;

events {
    worker_connections 1024;
}

http {
    client_max_body_size 50M;

    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" '
        '"$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;

    sendfile on; # 응답을 보낼 때 user 영역 buffer가 아닌, kernel file buffer를 사용

    keepalive_timeout 65;

    upstream docker-nginx {
        server nginx:80;
    }

    server {
        listen 80;

        listen [::]:80;

        server_name "도메인 주소";

        location ~ /\.well-known/acme-challenge {
            allow all;

            root /usr/share/nginx/html;

            try_files $uri =404;
        }

        location / {
            return 301 https://$server_name$request_uri;
        }
    }

    server {
        listen 443 ssl;

        listen [::]:443 ssl;

        server_name jourgeois.com www.jourgeois.com;

        ssl_certificate /etc/letsencrypt/live/"도메인 주소"/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/"도메인 주소"/privkey.pem;

        # include /etc/letsencrypt/options-ssl-nginx.conf; # 보안 강화를 위한 옵션 추가
        # ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # 보안 강화를 위한 옵션 추가

        location /img {
            proxy_pass http://springboot:8080;

```

```

    proxy_redirect      off;

    rewrite ^/(.*)$ /$1 break;

    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-NginX-Proxy true;
}

location /api {

    proxy_pass          http://springboot:8080;

    proxy_redirect      off;

    rewrite ^/api/(.*)$ /$1 break;

    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-NginX-Proxy true;
}

location / {

    proxy_pass          http://docker-nginx;

    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;
}
}

```

### 3.5.2 default.conf ( ./nginx/default\_ssl.conf )

```

user nginx;
worker_processes auto;

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

events {
    worker_connections 1024;
}

http {
    client_max_body_size 50M;

    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" '
        '"$http_user_agent" "$http_x_forwarded_for"';
    access_log /var/log/nginx/access.log main;
    sendfile on; # 응답을 보낼 때 user 영역 buffer가 아닌, kernel file buffer를 사용
    keepalive_timeout 65;

    upstream docker-nginx {
        server nginx:80;
    }

    server {
        listen 80;

```

```

        location /img {
            proxy_pass            http://springboot:8080;
            proxy_redirect        off;

            rewrite ^/(.*)$ /$1 break;

            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-NginX-Proxy true;
        }

        location /api {
            proxy_pass            http://springboot:8080;
            proxy_redirect        off;

            rewrite ^/api/(.*)$ /$1 break;

            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-NginX-Proxy true;
        }

        location / {
            proxy_pass            http://docker-nginx;
            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;
        }
    }
}

```

### 3.5.3 MySQL 설정 - custom.conf

```

[mysqld]

skip-character-set-client-handshake

character-set-server      = utf8mb4

collation-server          = utf8mb4_unicode_ci

default-time-zone         = '+9:00'

[client]

default-character-set      = utf8mb4

[mysql]

default-character-set      = utf8mb4

[mysqldump]

default-character-set      = utf8mb4

```

## 3.6 AWS S3 Bucket – config, access, policy

1. IAM 계정을 만들고 key를 발급 받는다.

- 해당 키는 Springboot project에서 버킷에 대한 수정, 삭제, 조회를 위해 필요함

2. 버킷 설정

- 퍼블릭 액세스 차단 모두 해제



## 퍼블릭 액세스 차단 편집(버킷 설정) 정보

### 퍼블릭 액세스 차단(버킷 설정)

퍼블릭 액세스는 ACL(액세스 제어 목록), 버킷 정책, 액세스 지정 정책 또는 모두를 통해 버킷 및 객체에 부여됩니다. 모든 S3 버킷 및 객체에 대한 퍼블릭 액세스가 차단되었는지 확인하려면 [모든 퍼블릭 액세스 차단]을 활성화합니다. 이 설정은 이 버킷 및 해당 액세스 지정에만 적용됩니다. AWS에서는 [모든 퍼블릭 액세스 차단]을 활성화하도록 권장하지만, 이 설정을 적용하기 전에 퍼블릭 액세스가 없어도 애플리케이션이 올바르게 작동하는지 확인합니다. 버킷 또는 내부 객체에 어느 정도 수준의 퍼블릭 액세스가 필요한 경우 특정 스토리지 사용 사례에 맞게 아래 개별 설정을 사용자 지정할 수 있습니다. [자세히 알아보기](#)

#### ☐ 모든 퍼블릭 액세스 차단

이 설정을 활성화하면 아래 4개의 설정을 모두 활성화한 것과 같습니다. 다음 설정 각각은 서로 독립적입니다.

- ☐ 새 ACL(액세스 제어 목록)을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단  
S3은 새로 추가된 버킷 또는 객체에 적용되는 퍼블릭 액세스 권한을 차단하며, 기존 버킷 및 객체에 대한 새 퍼블릭 액세스 ACL 생성을 금지합니다. 이 설정은 ACL을 사용하여 S3 리소스에 대한 퍼블릭 액세스를 허용하는 기존 권한을 변경하지 않습니다.
- ☐ 임의의 ACL(액세스 제어 목록)을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단  
S3은 버킷 및 객체에 대한 퍼블릭 액세스를 부여하는 모든 ACL을 무시합니다.
- ☐ 새 퍼블릭 버킷 또는 액세스 지정 정책을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단  
S3은 버킷 및 객체에 대한 퍼블릭 액세스를 부여하는 새 버킷 및 액세스 지정 정책을 차단합니다. 이 설정은 S3 리소스에 대한 퍼블릭 액세스를 허용하는 기존 정책을 변경하지 않습니다.
- ☐ 임의의 퍼블릭 버킷 또는 액세스 지정 정책을 통해 부여된 버킷 및 객체에 대한 퍼블릭 및 교차 계정 액세스 차단  
S3은 버킷 및 객체에 대한 퍼블릭 액세스를 부여하는 정책을 사용하는 버킷 또는 액세스 지정에 대한 퍼블릭 및 교차 계정 액세스를 무시합니다.

- 버킷 정책 설정

### 버킷 정책

JSON으로 작성된 버킷 정책은 버킷에 저장된 객체에 대한 액세스 권한을 제공합니다. 버킷 정책은 다른 계정이 소유한 객체에는 적용되지 않습니다.

```
{
  "Version": "2012-10-17",
  "Id": "Policy1658457927034",
  "Statement": [
    {
      "Sid": "Stmt1658457915425",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "s3:DeleteObject",
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": "arn:aws:s3:::jourgeois-profile-image/*"
    }
  ]
}
```

## 4. 배포 상세 내용

\*\* 3. 빌드 상세 내용 과정이 필수적으로 선행 되어야 한다.

### <수동 배포>

1. Repository를 clone 한 후에 ./backend 디렉토리에서 gradlew build 실행
2. ./clear\_con.sh을 실행하여 현재 올라가 있는 docker container를 삭제(포트 충돌을 방지하기 위함, 다른 기동 중이 서비스가 있다면 dockerfile 혹은 docker-compose.yml을 수정하여 포트 변경 작업 필요)
3. docker-compose build && docker-compose up

### <자동 배포>


#### 4.1 <Jenkins>

1. Jenkins 설치(배포 서버 로컬에 설치)
2. 새 자유 프로젝트 생성

**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(형상관리)으로 묶인 당신의 프로젝트를 빌드할 것이고, 소프트웨어 빌드보다 다른 어떤 것에 자주 사용될 수 있습니다.

### 3. Gitlab repository 설정

- repository url을 입력하고, credential을 입력한다.

**Git**

Repositories

Repository URL

Please enter Git repository.

Credentials

- none -

+ Add

Jenkins

Refspec

### 4. (credential이 없는 경우) Gitlab 계정과 비밀번호를 입력하고 credential을 생성한다.

**Jenkins Credentials Provider: Jenkins**

**Add Credentials**

Domain

Global credentials (unrestricted)

Kind

Username with password

Scope

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

Username

☐ Treat username as secret

### 5. 빌드할 branch를 설정한다.

**Branches to build**

Branch Specifier (blank for 'any')

\*/master

Add Branch

### 6. 빌드 유발 설정 - Gitlab webhook 발생시 빌드하도록 한다.

☒ Build when a change is pushed to GitLab. GitLab webhook URL: [http://13.209.206.237:8080/project/new\\_project](http://13.209.206.237:8080/project/new_project) ?

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

7. 빌드 유발 설정 - 고급 탭에서 gitlab webhook과 연동하기 위한 키를 발급받는다.

8. 빌드 스크립트 작성 - build 탭에 Execute Shell을 선택하고 아래 script를 작성하여 입력한다.

Secret token ?

Generate

<GitLab>

8. 6 에서의 webhook URL과 7 에서의 Secret token을 입력한다.

Search page

**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

Push to the repository

<Jenkins>

9. 빌드 스크립트 작성

```
sudo chmod 755 ./backend/gradlew
sudo chmod 755 ./clear_con.sh
cd ./backend
sudo gradlew build

cd ..

# 기존에 올라가 있는 docker 컨테이너를 모두 삭제
sudo ./clear_con.sh

sudo docker-compose build

sudo docker-compose up -d

-----
* Jenkins 계정을 sudoer에 추가해주어야함
* clear_con.sh => docker rm -f $(docker ps -a -q) 2 >dev/null
```

## 5. Ignore 파일

frontend : .env (최상단 위치)

backend : application.yml, serviceAccountKey.json, aws.properties, email.properties