

# 포팅 매뉴얼

## 포팅 매뉴얼

### 1. Gitlab 소스 클론 이후 빌드 및 배포할 수 있도록 정리한 문서

#### 1.1 JVM (Java Virtual Machine)

- 종류와 설정 값:
  - JVM: OpenJDK 17
  - 설정 값: `JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64`

#### 1.2 웹서버

- 종류와 설정 값:
  - 웹서버: Nginx
  - 설정 파일 경로: `/etc/nginx/conf.d/default.conf`
  - 주요 설정:

```
server {  
    listen 3000;  
  
    location / {  
        root /usr/share/nginx/html;  
        index index.html index.htm;  
        try_files $uri $uri/ /index.html;  
    }  
}
```

#### 1.3 WAS (Web Application Server)

- 종류와 설정 값:
  - WAS: Spring Boot
  - Spring Boot 버전: 3.2.4

◦ 주요 설정:

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '3.2.4'
    id 'io.spring.dependency-management' version '1.1.4'
    id 'org.asciidoctor.jvm.convert' version '3.3.2'
    id "org.sonarqube" version "5.0.0.4638"
}

group = 'com.potless'
version = '0.0.1-SNAPSHOT'

java {
    sourceCompatibility = '17'
}

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-data-redis'
    implementation 'org.springframework.boot:spring-boot-starter-security'
    implementation 'org.springframework.boot:spring-boot-starter-validation'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    runtimeOnly 'com.mysql:mysql-connector-j'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    testImplementation 'org.springframework.restdocs:spring-restdocs-mockmvc'
    testImplementation 'org.springframework.security:spr
```

```

ing-security-test'
    implementation 'org.springframework.boot:spring-boot
-starter-logging'
    asciidoctorExt 'org.springframework.restdocs:spring-
restdocs-asciidoctor'
    testImplementation 'org.springframework.restdocs:spr
ing-restdocs-mockmvc'
    compileOnly 'org.projectlombok:lombok'
    annotationProcessor 'org.projectlombok:lombok'
    testCompileOnly 'org.projectlombok:lombok'
    testAnnotationProcessor 'org.projectlombok:lombok'
    implementation 'com.querydsl:querydsl-jpa:5.1.0:jaka
rta'
    annotationProcessor "com.querydsl:querydsl-apt:${dep
endencyManagement.importedProperties['querydsl.version
']}:jakarta"
    annotationProcessor "jakarta.annotation:jakarta.anno
tation-api"
    annotationProcessor "jakarta.persistence:jakarta.per
sistence-api"
    implementation 'org.springframework.boot:spring-boot
-starter-security'
    compileOnly group: 'io.jsonwebtoken', name: 'jjwt-ap
i', version: '0.11.2'
    runtimeOnly group: 'io.jsonwebtoken', name: 'jjwt-im
pl', version: '0.11.2'
    runtimeOnly group: 'io.jsonwebtoken', name: 'jjwt-ja
ckson', version: '0.11.2'
    implementation 'org.springframework.boot:spring-boot
-starter-mail'
    implementation 'com.amazonaws:aws-java-sdk-s3:1.12.7
06'
    implementation 'org.springframework.boot:spring-boot
-starter-webflux'
    implementation 'org.springdoc:springdoc-openapi-star
ter-webmvc-ui:2.0.2'
    implementation 'com.uber:h3:3.7.2'
}

```

```

tasks.named('test') {
    outputs.dir snippetsDir
    useJUnitPlatform()
}

tasks.named('asciidoctor') {
    inputs.dir snippetsDir
    configurations 'asciidoctorExt'

    sources {
        include("**/index.adoc")
    }
    baseDirFollowsSourceFile()
    dependsOn test
}

bootJar {
    dependsOn asciidoctor
    from("${asciidoctor.outputDir}") {
        into 'static/docs'
    }
}

clean {
    delete file('src/main/generated')
}

```

## 1.4 서버 상세 정보

- **EC2 인스턴스 1 (xlarge):**
  - 인스턴스 유형: EC2 xlarge (Lightsail)
  - VCPU: 4
  - RAM: 16GB
  - 스토리지: gp3 - 320GB
  - 루트 디바이스 이름: `/dev/sda1`

- 루트 디바이스 유형: EBS
- EBS 최적화: 활성화
- **EC2 인스턴스 2 (c6i.2xlarge):**
  - 인스턴스 유형: c6i.2xlarge
  - 루트 디바이스 이름: `/dev/sda1`
  - 루트 디바이스 유형: EBS
  - EBS 최적화: 활성화

## 2. 프로젝트에서 사용하는 외부 서비스 정보를 정리한 문서

### 2.1 Kakao API

- **사용 목적:** 소셜 로그인, 지도 서비스, 메시지 전송 등
- **환경 변수 설정:**

```
ARG VITE_KAKAO_APP_KEY
ARG VITE_KAKAO_REST_API_KEY

ENV VITE_KAKAO_APP_KEY=${VITE_KAKAO_APP_KEY} \
    VITE_KAKAO_REST_API_KEY=${VITE_KAKAO_REST_API_KEY}
```

- **사용 예시:**

```
const kakaoAppKey = process.env.VITE_KAKAO_APP_KEY;
const kakaoRestApiKey = process.env.VITE_KAKAO_REST_API_KEY;

Kakao.init(kakaoAppKey);
```

- **필요한 정보:**
  - Kakao Developers 계정 생성
  - 애플리케이션 등록 후 앱 키(App Key) 발급
  - Redirect URI 설정

### 2.2 Weather API

- **사용 목적:** 날씨 정보 제공
- **환경 변수 설정:**

```
RG VITE_WEATHER_API_KEY

ENV VITE_WEATHER_API_KEY=${VITE_WEATHER_API_KEY}
```

- **사용 예시:**

```
const weatherApiKey = process.env.VITE_WEATHER_API_KEY;
const weatherApiUrl = `https://api.weather.com/v3/wx/forecast/daily/5day?apiKey=${weatherApiKey}&language=en-US&format=json`;

fetch(weatherApiUrl)
  .then(response => response.json())
  .then(data => console.log(data));
```

- **필요한 정보:**
  - Weather API 제공업체 계정 생성
  - API Key 발급

## 2.3 기타 외부 서비스

- **포트 클라우드:** AWS S3, CloudFront 등
  - **환경 변수 설정:**

```
G VITE_SERVICE_URL

ENV VITE_SERVICE_URL=${VITE_SERVICE_URL}
```

- **사용 예시:**

```
const serviceUrl = process.env.VITE_SERVICE_URL;
console.log(`Service URL: ${serviceUrl}`);
```

- **필요한 정보:**

- AWS 계정 생성
- S3 버킷 및 CloudFront 배포 설정
- 접근 키 및 비밀 키 발급

## 2.4 Dockerfile 예시

```
# Build stage
FROM node:alpine as build-stage

ARG VITE_SERVICE_URL
ARG VITE_KAKAO_APP_KEY
ARG VITE_KAKAO_REST_API_KEY
ARG VITE_WEATHER_API_KEY

ENV VITE_SERVICE_URL=${VITE_SERVICE_URL} \
    VITE_KAKAO_APP_KEY=${VITE_KAKAO_APP_KEY} \
    VITE_KAKAO_REST_API_KEY=${VITE_KAKAO_REST_API_KEY} \
    VITE_WEATHER_API_KEY=${VITE_WEATHER_API_KEY}

RUN echo "VITE_SERVICE_URL: $VITE_SERVICE_URL"
RUN echo "VITE_KAKAO_APP_KEY: $VITE_KAKAO_APP_KEY"
RUN echo "VITE_KAKAO_REST_API_KEY: $VITE_KAKAO_REST_API_KEY"
RUN echo "VITE_WEATHER_API_KEY: $VITE_WEATHER_API_KEY"

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

RUN npm run build

# Production stage
FROM nginx:alpine as production-stage
```

```
COPY ./nginx/default.conf /etc/nginx/conf.d/default.conf
```

```
COPY --from=build-stage /app/dist /usr/share/nginx/html
```

## 2.5 외부 서비스 정보 요약

- **Kakao API:**
  - **사용 목적:** 소셜 로그인, 지도 서비스, 메시지 전송
  - **필요한 정보:** Kakao Developers 계정, 앱 키, Redirect URI 설정
  - **환경 변수:** `VITE_KAKAO_APP_KEY`, `VITE_KAKAO_REST_API_KEY`
- **Weather API:**
  - **사용 목적:** 날씨 정보 제공
  - **필요한 정보:** Weather API 계정, API Key
  - **환경 변수:** `VITE_WEATHER_API_KEY`
- **포트 클라우드:**
  - **사용 목적:** AWS S3, CloudFront 등
  - **필요한 정보:** AWS 계정, S3 버킷, CloudFront 배포, 접근 키 및 비밀 키
  - **환경 변수:** `VITE_SERVICE_URL`

## 3. DB 덤프 파일 최신본

## 4. 시연 시나리오

- 시연 순서에 따른 site 화면별, 실행법(클릭 위치 등) 상세 설명

## MySQL EC2 서버 연결 정보

### 기본 정보

- **호스트 이름 (hostname):** `mine702-db.cj6oog44srac.ap-northeast-2.rds.amazonaws.com`
- **포트 (port):** `3306`
- **사용자 이름 (username):** `mine702`
- **비밀번호 (password):** `633ehddbbs`



- 인스턴스 유형: `db.t3.micro`

## MySQL 연결 설정 예시

아래는 MySQL 워크벤치 또는 다른 MySQL 클라이언트를 사용하여 연결할 수 있는 설정 예시입니다.

## MySQL Workbench 설정

1. MySQL Workbench 열기.
2. 새 연결 추가:
  - **Connection Name:** 원하는 이름 입력 (예: `mine702-db-connection`)
  - **Hostname:** `mine702-db.cj6oog44srac.ap-northeast-2.rds.amazonaws.com`
  - **Port:** `3306`
  - **Username:** `mine702`
  - **Password:** `633ehddb5`
3. 테스트 연결:
  - **Test Connection** 버튼을 클릭하여 연결 확인.
4. 연결 저장:
  - 연결이 성공하면 **OK**를 눌러 연결 저장.

## AWS S3 및 IAM 인증 정보

### 기본 정보

- **IAM ARN:** `arn:aws:iam::767397822380:user/mine`
- **액세스 키 ID (Access Key ID):** `AKIA3FLDZP6WNF5CGZGX`

## c6 인스턴스 설정

```
ssh -i ~/AI.pem ubuntu@54.180.235.189
```

AI.pem

이 인스턴스에서는 Nginx Proxy Manager, MariaDB, 그리고 FastAPI를 실행합니다.

## Docker Compose 파일

```
version: "3.8"
services:
  app:
    image: "jc21/nginx-proxy-manager:latest"
    restart: unless-stopped
    ports:
      - "80:80"
      - "60081:81"
      - "443:443"
    environment:
      # Mysql/Maria connection parameters:
      DB_MYSQL_HOST: "db"
      DB_MYSQL_PORT: 3306
      DB_MYSQL_USER: "npm"
      DB_MYSQL_PASSWORD: "npm"
      DB_MYSQL_NAME: "npm"
    volumes:
      - ./data:/data
      - ./letsencrypt:/etc/letsencrypt
    depends_on:
      - db

  db:
    image: "jc21/mariadb-aria:latest"
    restart: unless-stopped
    environment:
      MYSQL_ROOT_PASSWORD: "npm"
      MYSQL_DATABASE: "npm"
      MYSQL_USER: "npm"
      MYSQL_PASSWORD: "npm"
      MARIADB_AUTO_UPGRADE: "1"
    volumes:
      - ./data/mysql:/var/lib/mysql

  fastapi:
```

```
image: mine0702/potless-fastapi
restart: unless-stopped
container_name: app_fastapi
stdin_open: true
expose:
  - "8000"
deploy:
  resources:
    limits:
      cpus: "0"
      memory: "0"
    reservations:
      cpus: "0"
      memory: "0"
```

## 포트 설정

- 80: HTTP
- 443: HTTPS
- 60081: Nginx Proxy Manager Admin UI
- 3306: MariaDB
- 8000: FastAPI (내부 노출)

## 설정 및 실행 절차

### 1. 디렉토리 이동 및 권한 설정:

```
cd /path/to/docker-compose-directory
sudo chmod -R 777 .
```

### 2. Docker Compose 실행:

```
docker-compose up -d
```

### 3. MariaDB 설정:

```
docker exec -it $(docker ps -q -f name=db) mysql -u root  
-
```

#### 4. MariaDB 내부에서:

```
CREATE USER 'npm'@'%' IDENTIFIED BY 'npm';  
GRANT ALL PRIVILEGES ON *.* TO 'npm'@'%';  
CREATE DATABASE npm;  
FLUSH PRIVILEGES
```

### 보안 그룹 설정 (AWS 콘솔에서)

- 인바운드 규칙:
  - HTTP (포트 80)
  - HTTPS (포트 443)
  - Custom TCP (포트 60081)
  - MySQL/Aurora (포트 3306) - 필요한 경우 특정 IP로 제한
  - Custom TCP (포트 8000) - 필요 시 내부 네트워크에서만 접근 가능하도록 설정

### xlarge 인스턴스 설정

```
ssh -i ~/K10B106T.pem ubuntu@k10b106.p.ssafy.io
```

K10B106T.pem

이 인스턴스에서는 Nginx Proxy Manager, MariaDB, 프론트엔드, 그리고 백엔드 서비스를 실행합니다.

### Docker Compose 파일

```
version: "3.8"  
services:  
  app:
```

```
image: "jc21/nginx-proxy-manager:latest"
restart: unless-stopped
ports:
  - "80:80"
  - "60081:81"
  - "443:443"
environment:
  # Mysql/Maria connection parameters:
  DB_MYSQL_HOST: "db"
  DB_MYSQL_PORT: 3306
  DB_MYSQL_USER: "npm"
  DB_MYSQL_PASSWORD: "npm"
  DB_MYSQL_NAME: "npm"
volumes:
  - ./data:/data
  - ./letsencrypt:/etc/letsencrypt
depends_on:
  - db
```

```
db:
  image: "jc21/mariadb-aria:latest"
  restart: unless-stopped
  environment:
    MYSQL_ROOT_PASSWORD: "npm"
    MYSQL_DATABASE: "npm"
    MYSQL_USER: "npm"
    MYSQL_PASSWORD: "npm"
    MARIADB_AUTO_UPGRADE: "1"
  volumes:
    - ./data/mysql:/var/lib/mysql
```

```
frontend:
  image: mine0702/potless-frontend
  restart: unless-stopped
  container_name: app_frontend
  expose:
    - "3000"
  stdin_open: true
```

```
backend:
  image: mine0702/potless-backend
  restart: unless-stopped
  container_name: app_backend
  stdin_open: true
  expose:
    - "7080"
```

## 포트 설정

- 80: HTTP
- 443: HTTPS
- 60081: Nginx Proxy Manager Admin UI
- 3306: MariaDB
- 3000: 프론트엔드 (내부 노출)
- 7080: 백엔드 (내부 노출)

## 설정 및 실행 절차

### 1. 디렉토리 이동 및 권한 설정:

```
cd /path/to/docker-compose-directory
sudo chmod -R 777
```

### 2. Docker Compose 실행:

```
docker-compose up -
```

### 3. MariaDB 설정:

```
docker exec -it $(docker ps -q -f name=db) mysql -u root
-p
```

### 4. MariaDB 내부에서:

```
CREATE USER 'npm'@'%' IDENTIFIED BY 'npm';  
GRANT ALL PRIVILEGES ON *.* TO 'npm'@'%';  
CREATE DATABASE npm;  
FLUSH PRIVILEGES;
```

## 보안 그룹 설정 (AWS 콘솔에서)

- 인바운드 규칙:
  - HTTP (포트 80)
  - HTTPS (포트 443)
  - Custom TCP (포트 60081)
  - MySQL/Aurora (포트 3306) - 필요한 경우 특정 IP로 제한
  - Custom TCP (포트 3000) - 필요 시 내부 네트워크에서만 접근 가능하도록 설정
  - Custom TCP (포트 7080) - 필요 시 내부 네트워크에서만 접근 가능하도록 설정