포팅 매뉴얼

개발환경

FrontEnd

- Node.js: `18.15.0`
- npm: `8.19.3`
- React: `18.2.0`

DevOps

- Docker: `23.0.1`
- Jenkins: `2.387.1`
- Nginx: `nginx/1.18.0`

Server

- AWS EC2: `ubuntu 20.04`
- IntelliJ: `IDEA 2022.3.1`
- SpringBoot: `2.7.9`
- JDK: `OpenJDK 11.0.17`

Database

- MySQL: `8.0.32`
- Redis: `7.0.10`

관리

- GitLab
- Jira
- Notion
- Slack

프로젝트 빌드 및 배포

AWS EC2 내 Docker 설치

Install Docker Engine on Ubuntu

Jumpstart your client-side server applications with Docker Engine on Ubuntu. This guide details prerequisites and multiple methods to install.

https://docs.docker.com/engine/install/ubuntu/

1. 패키지 업데이트 진행

```
sudo apt-get update
```

2. 필요 패키지 설치 (복사하면 바로 안됨…)

```
sudo apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

3. 도커의 Official GPG Key를 등록

```
sudo mkdir -p /etc/apt/keyrings

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

4. stable repository 등록

```
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
```

5. 도커와 도커 컴포즈 설치

```
# 도커 설치
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin


# 도커 컴포즈 설치
sudo apt install docker-compose
```

6. 도커와 도커 컴포즈 확인

```
# 도커 설치 확인
sudo docker -v

# 도커 컴포즈 설치 확인
sudo docker-compose -v
```

AWS EC2 내 Jenkins 설치

1. docker-compose.yml 작성 (스페이스바 2개)

```
# docker-compose.yml
version: '3.4'

services:
  jenkins:
    image: jenkins/jenkins:lts
    user: root
    restart: always
    container_name: jenkins
    volumes:
      - ./jenkins:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
    ports:
      - 8080:8080
```

2. 도커 컴포즈 파일로 컨테이너 띄우기

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

3. 젠킨스 내에서 docker command를 실행시켜야 하기 때문에 도커 설치

```
# 젠킨스 접속
sudo docker exec -it jenkins bash

# 젠킨스 안에 도커 설치
curl -fsSL https://get.docker.com -o get-docker.sh
sh get-docker.sh
```

Jenkins 설정 및 플러그인 설치

1. 젠킨스 admin 비밀번호 확인
   - 방법 1 : Log 확인

     ```
     sudo docker logs jenkins
     ```

   - 방법 2 : /var/jenkins_home/secrets/initialAdminPassword 파일 확인

     ```
     // jenkins container ID 찾는 방법 : docker ps

     sudo docker exec {jenkins container ID} cat /var/jenkins_home/secrets/initialAdminPassword
     ```

2. 젠킨스에 접속해서 비밀번호 입력 후 기본 플러그인 설치

```
http://k8a508.p.ssafy.io:8080/
```

3. 젠킨스에 접속할 Admin 계정 생성 (Create First Admin User)
   - 계정명 : rollwrite
   - 비밀번호 : rollwrite123!

4. 추가 플러그인 설치

- GitLab `1.7.9`
- NodeJS `1.6.0`
- Generic Webhook Trigger Plugin `1.86.2`

Jenkins 아이템 생성과 WebHook 설정

> 💡 개발 서버와 운영 서버를 분리

1. Credential 생성
   - Username : gitlab의 사용자 id
   - Password : gitlab의 사용자 password
   - ID : Credentials를 구분하는 ID

2. 아이템 생성
   - FE (Freestyle project)
   - BE (Freestyle project)
   - release (Freestyle project)

3. 소스 코드 관리
   - Repositories
     - Repository URL : https://lab.ssafy.com/s08-final/S08P31A508.git
     - Credentials : 위에서 만든 Credential로 지정
   - Branches to build
     - Branch Specifier
       - FE : */fe
       - BE : */be
       - release : */develop

4. 빌드 유발
   - **Build when a change is pushed to GitLab. GitLab webhook URL:**
     - Push Events 선택
   - Allowed branches
     - Filter branches by name
       - Include
         - FE : fe
         - BE : be
         - release : develop

5. Gitlab에서 Webhooks 설정
   - Gitlab → Settings → Webhooks로 이동
   - URL

    - http://k8a508.p.ssafy.io:8080/project/FE
    - http://k8a508.p.ssafy.io:8080/project/BE
    - http://k8a508.p.ssafy.io:8080/project/release
- Secret token : 젠킨스에서 만든 Secret token

Jenkins 빌드 설정

FE 설정

1. /var/jenkins_home/env/에 .env 추가
2. 젠킨스 접속 → Jenkins 관리 → Global Tool Configuration으로 이동
3. NodeJS intallations 추가
4. Version : NodeJS `18.12.1`
5. FE → Configuration으로 이동
6. Build Steps
    - Execute NodeJS script : 방금 생성한 NodeJS Installation 추가
    - Execute shell

```
// .env 옮겨주는 과정
cp /var/jenkins_home/env/.env ${WORKSPACE}/frontend/

cd frontend
npm install
CI=false npm run build

docker compose up --build -d
```

BE 설정

1. 젠킨스 서버 접속

```
sudo docker exec -it jenkins bash
```

2. /var/jenkins_home/env 디렉토리에 application.yml를 생성

    💡 application.yml를 gitignore에 추가했기 때문에 따로 서버에 저장해주는 과정이다

```
cd var/jenkins_home/env

vim application.yml
```

```
# application.yml
server:
  port: 8081
  servlet:
    context-path: /api

spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://db-mysql:[PORT]/[DATABASE]?serverTimezone=Asia/Seoul&characterEncoding=UTF-8&useUnicode=true
    username: [USERNAME]
    password: [PASSWORD]

  jpa:
    show-sql: true
```

```
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
        format_sql: true

  mvc:
    pathmatch:
      matching-strategy: ant_path_matcher

  logging:
    file:
      name: ./test.log
    level:
      root: debug
      org:
        springframework:
          web: debug
          boot: debug
          security: debug

  # multipart
  servlet:
    multipart:
      max-file-size: 20MB
      max-request-size: 20MB

  # redis
  redis:
    host: db-redis
    port: [PORT]
    password: [PASSWORD]

  # batch
  batch:
    job:
      names: question
    jdbc:
      initialize-schema: always

  # .yml 파일 include
  profiles:
    include:
      - auth-key

# feign
feign:
  client:
    config:
      default:
        connect-timeout: 100000000
        read-timeout: 100000000

# invite-url
inviteUrl : https://k8a508.p.ssafy.io/join/

# chat-gpt
openai-service:
  api-key:[GPT_API_KEY]
  #  gpt-model: gpt-3.5-turbo
  http-client:
    read-timeout: 3000
    connect-timeout: 3000
  urls:
    base-url: https://api.openai.com/v1
    chat-url: /chat/completions
```

```
# application-auth-key.yml
# jwt
jwt:
  secret:
    atk: [ATK]
    rtk: [RTK]
  expiration:
    atk: 86400000
    rtk: 864000000

# kakao Login
auth:
  kakao:
    redirect-uri: https://k8a508.p.ssafy.io/oauth
    rest-key: [REST-KEY]
    secret-key: [SECRET-KEY]
```

```
# FCM
fcm:
  key:
    path: /rollwrite-1c979-firebase-adminsdk-4e75m-de6b0ec02f.json
    url: https://fcm.googleapis.com/v1/projects/rollwrite-1c979/messages:send

# Security Pass Uri
security:
  passuri: /api/auth/kakao/login, /api/auth/reissue
```

```
{
  "type": "service_account",
  "project_id": "rollwrite-1c979",
  "private_key_id": "${PRIVATE-KEY-ID}",
  "private_key": "-----BEGIN PRIVATE KEY-----\${PRIVATE-KEY}\n-----END PRIVATE KEY-----\n",
  "client_email": "firebase-adminsdk-4e75m@rollwrite-1c979.iam.gserviceaccount.com",
  "client_id": "118386961724431419530",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/firebase-adminsdk-4e75m%40rollwrite-1c979.iam.gservic
}
```

3. docker-compose.yml와 같은 위치에 .env 생성

```
MYSQL_ROOT_PASSWORD=[MYSQL_ROOT_PASSWORD]
MYSQL_ROOT_HOST=[MYSQL_ROOT_HOST]
MYSQL_USER=[MYSQL_USER]
MYSQL_PASSWORD=[MYSQL_PASSWORD]
REDIS_PASSWORD=[REDIS_PASSWORD]
```

4. BE → Configuration으로 이동
5. Build Steps
   - Execute shell

```
// application.yml 옮겨주는 과정
cp /var/jenkins_home/env/application.yml ${WORKSPACE}/backend/src/main/resources
cp /var/jenkins_home/env/application-auth-key.yml ${WORKSPACE}/backend/src/main/resources
cp /var/jenkins_home/env/rollwrite-1c979-firebase-adminsdk-4e75m-de6b0ec02f.json ${WORKSPACE}/backend/src/main/resources

cd backend
chmod +x gradlew
./gradlew --stacktrace clean build -x test

docker compose up --build -d
```

release 설정

1. /var/jenkins_home/env/release/에 .env 추가
2. docker-compose.yml와 같은 위치에 .env 생성

```
REDIS_PASSWORD=[REDIS_PASSWORD]
```

3. release → Configuration으로 이동
4. Build Steps
   - Execute NodeJS script : 방금 생성한 NodeJS Installation 추가
   - Execute shell

```
// .env, application.yml 옮겨주는 과정
cp /var/jenkins_home/env/release/.env ${WORKSPACE}/frontend/
```

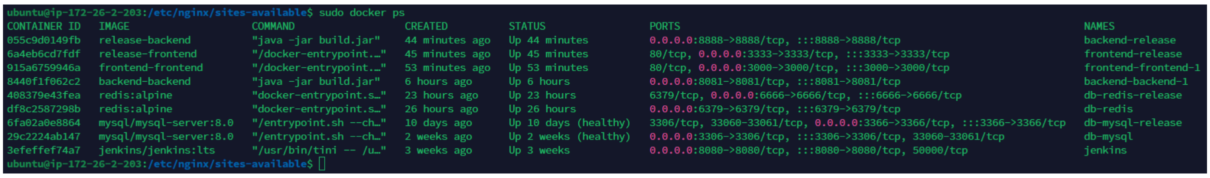
```
cp /var/jenkins_home/env/release/application.yml ${WORKSPACE}/backend/src/main/resources
cp /var/jenkins_home/env/release/application-auth-key.yml ${WORKSPACE}/backend/src/main/resources
cp /var/jenkins_home/env/release/rollwrite-1c979-firebase-adminsdk-4e75m-de6b0ec02f.json ${WORKSPACE}/backend/src/main/resour

cd frontend
npm install
CI=false npm run build

cd ../backend
chmod +x gradlew
./gradlew --stacktrace clean build -x test

cd ..
docker compose up --build -d
```

생성된 컨테이너 확인



```
ubuntu@ip-172-26-2-203:/etc/nginx/sites-available$ sudo docker ps
CONTAINER ID   IMAGE                   COMMAND                  CREATED          STATUS                  PORTS                                                        NAMES
055c9d0149fb   release-backend         "java -jar build.jar"    44 minutes ago   Up 44 minutes           0.0.0.0:8888->8888/tcp, :::8888->8888/tcp                    backend-release
6a4eb6cd7fdf   release-frontend        "/docker-entrypoint.…"   45 minutes ago   Up 45 minutes           80/tcp, 0.0.0.0:3333->3333/tcp, :::3333->3333/tcp            frontend-release
915a6759946a   frontend-frontend       "/docker-entrypoint.…"   53 minutes ago   Up 53 minutes           80/tcp, 0.0.0.0:3000->3000/tcp, :::3000->3000/tcp            frontend-frontend-1
8440f1f062c2   backend-backend         "java -jar build.jar"    6 hours ago      Up 6 hours              0.0.0.0:8081->8081/tcp, :::8081->8081/tcp                    backend-backend-1
408379e43fea   redis:alpine            "docker-entrypoint.s…"   23 hours ago     Up 23 hours             6379/tcp, 0.0.0.0:6666->6666/tcp, :::6666->6666/tcp          db-redis-release
df8c2587298b   redis:alpine            "docker-entrypoint.s…"   26 hours ago     Up 26 hours             0.0.0.0:6379->6379/tcp, :::6379->6379/tcp                    db-redis
6fa02a0e8864   mysql/mysql-server:8.0  "/entrypoint.sh --ch…"   10 days ago      Up 10 days (healthy)    3306/tcp, 33060-33061/tcp, 0.0.0.0:3366->3366/tcp, :::3366->3366/tcp   db-mysql-release
29c2224ab147   mysql/mysql-server:8.0  "/entrypoint.sh --ch…"   2 weeks ago      Up 2 weeks (healthy)    0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060-33061/tcp   db-mysql
3efeffef74a7   jenkins/jenkins:lts     "/usr/bin/tini -- /u…"   3 weeks ago      Up 3 weeks              0.0.0.0:8080->8080/tcp, :::8080->8080/tcp, 50000/tcp         jenkins
ubuntu@ip-172-26-2-203:/etc/nginx/sites-available$
```

Nginx & SSL 설정

1. 가비아에서 도메인 구매 후 연결

[ec2 2탄] 가비아에서 도메인 구매 후 연결하기 & HTTPS 적용하기
https://ye5ni.tistory.com/131 [ec2 1탄] AWS EC2 임대 및 서버 구축해보기 1. AWS EC2 인스턴스 생성 자세한 과정은 밑에서 확인!!! ↓↓↓ 더보기 ✅ AWS(Amazon Web Service) EC2 뽀개기 1. AWS 회원 가입하기 https://ap-northeast-2.console.aws.amazon.com/console/home?region=ap-n.. ye5ni.tistory.com ↑↑↑ 위 내
https://ye5ni.tistory.com/132

2. Nginx 설정

```
sudo apt-get install nginx
```

3. 설치 확인

```
sudo nginx -v
```

4. Nginx 중지

```
sudo systemctl stop nginx
```

5. Let's Encrypt 설치

```
sudo apt-get install letsencrypt
```

6. 인증서 적용 및 .pem 키

```
sudo letsencrypt certonly --standalone -d [도메인]
```

7. 발급 경로 확인

```
cd /etc/letsencrypt/live/[도메인]
```

8. 이동 후 conf 파일 생성

```
cd /etc/nginx/sites-available
sudo vim rollwrite.conf
```

```
# rollwrite.conf
# 개발 서버 - k8a508.p.ssafy.io
server {
        location / {
                proxy_pass http://localhost:3000;
        }

        location /api {
                proxy_pass http://localhost:8081/api;
        }

  listen 443 ssl;
  server_name k8a508.p.ssafy.io;

  client_max_body_size 100M;

  # ssl 인증서 적용하기
  ssl_certificate /etc/letsencrypt/live/k8a508.p.ssafy.io/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/k8a508.p.ssafy.io/privkey.pem;

}

server {
    if ($host = k8a508.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80;
    server_name k8a508.p.ssafy.io;
      return 404; # managed by Certbot
}
```

```
# rollwrite-release.conf
# 운영 서버 - rollwrite.co.kr
server {
        location / {
                proxy_pass http://localhost:3333;
        }

        location /api {
                proxy_pass http://localhost:8888/api;
        }

  listen 443 ssl;
  server_name rollwrite.co.kr;

  client_max_body_size 100M;

  # ssl 인증서 적용하기
  ssl_certificate /etc/letsencrypt/live/rollwrite.co.kr/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/rollwrite.co.kr/privkey.pem;

}

server {
    if ($host = rollwrite.co.kr) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

    listen 80;
```

```
    server_name rollwrite.co.kr;
      return 404; # managed by Certbot
}
```

9. 파일 연동 및 테스트

```
sudo ln -s /etc/nginx/sites-available/rollwrite.conf /etc/nginx/sites-enabled/rollwrite.conf
sudo ln -s /etc/nginx/sites-available/rollwrite-release.conf /etc/nginx/sites-enabled/rollwrite-release.conf
sudo nginx -t
```

10. Nginx 재시작

```
sudo systemctl restart nginx
```

11. Nginx 상태 확인

```
sudo systemctl status nginx
```

docker-compose.yml 및 Dockerfile

1. FE

```
# Dockerfile
FROM nginx:stable-alpine
WORKDIR /app
RUN mkdir ./build
ADD ./build ./build
RUN rm /etc/nginx/conf.d/default.conf
COPY ./nginx.conf /etc/nginx/conf.d
CMD ["nginx", "-g", "daemon off;"]
```

```
# docker-compose.yml
version: "3.7"

services:
  frontend:
    build: .
    ports:
      - 3000:3000
    restart: always
    volumes:
      - /home/ubuntu/files:/app/build/rollwrite
```

```
# nginx.conf
server {
    listen 3000;
    location / {
        root /app/build;
        index index.html;
        try_files $uri $uri/ /index.html;
    }
}
```

2. BE

```
# Dockerfile
FROM openjdk:11-jre-slim
ARG JAR_FILE="build/libs/rollwrite-0.0.1-SNAPSHOT.jar"
```

```
COPY ${JAR_FILE} build.jar
ENTRYPOINT ["java","-jar","build.jar"]
```

```
# docker-compose.yml
version: "3.7"

services:
  db-mysql:
    container_name: db-mysql
    image: mysql/mysql-server:8.0
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_ROOT_HOST: ${MYSQL_ROOT_HOST}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
      MYSQL_DATABASE: 'rollwrite'
      TZ: Asia/Seoul
    restart: always
    volumes:
      - ./mysql/data:/var/lib/mysql --user 1000
    ports:
      - '3306:3306'
    command:
      - '--character-set-server=utf8mb4'
      - '--collation-server=utf8mb4_unicode_ci'

  db-redis:
    container_name: db-redis
    image: redis:alpine
    environment:
      - TZ=Asia/Seoul
    hostname: db-redis
    labels:
      - "name=db-redis"
      - "mode=standalone"
    ports:
      - 6379:6379
    command: redis-server --port 6379 --requirepass ${REDIS_PASSWORD} --appendonly yes --replica-read-only no

  backend:
    build: .
    environment:
      - TZ=Asia/Seoul
    ports:
      - 8081:8081
    restart: always
    volumes:
      - /home/ubuntu/files:/var/lib/rollwrite
```

3. release

```
# Dockerfile - front
FROM nginx:stable-alpine
WORKDIR /app
RUN mkdir ./build
ADD ./frontend/build ./build
RUN rm /etc/nginx/conf.d/default.conf
COPY ./nginx.conf /etc/nginx/conf.d
CMD ["nginx", "-g", "daemon off;"]
```

```
# Dockerfile - back
FROM openjdk:11-jre-slim
ARG JAR_FILE="build/libs/rollwrite-0.0.1-SNAPSHOT.jar"
COPY ${JAR_FILE} build.jar
ENTRYPOINT ["java","-jar","build.jar"]
```

```
# docker-compose.yml
version: "3.7"

services:
  db-redis:
    container_name: db-redis-release
    image: redis:alpine
    environment:
      - TZ=Asia/Seoul
    hostname: db-redis
    labels:
```

```
      - "name=db-redis"
      - "mode=standalone"
    ports:
      - 6666:6666
    command: redis-server --port 6666 --requirepass ${REDIS_PASSWORD} --appendonly yes --replica-read-only no

  backend:
    container_name: backend-release
    build: ./backend
    environment:
      - TZ=Asia/Seoul
    ports:
      - 8888:8888
    restart: always
    volumes:
      - /home/ubuntu/files-release:/var/lib/rollwrite

  frontend:
    container_name: frontend-release
    build: .
    ports:
      - 3333:3333
    restart: always
    volumes:
      - /home/ubuntu/files-release:/app/build/rollwrite
```

```
# nginx.conf
server {
    listen 3333;
    location / {
        root /app/build;
        index index.html;
        try_files $uri $uri/ /index.html;
    }
}
```