

Porting Manual

배포 환경 설정

서버

- AWS EC2(ubuntu 20.04 LTS)
-

Docker 설치

- Docker version 20.10.17

1. 오래된 버전 삭제

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

2. apt update 및 repository 설정

```
sudo apt-get update

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

3. Docker의 Official GPG Key 등록

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

4. stable repository 등록

```
echo \
"deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

5. Docker Engine 설치

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

6. 설치 후 버전 확인

```
docker --version
```

7. docker-compose 설치

```
sudo apt-get update
sudo apt-get install docker-compose-plugin

# 설치 확인
docker compose version
```

MySQL 설치

1. MySQL APT Repository 추가 및 패키지 다운로드

```
sudo wget https://dev.mysql.com/get/mysql-apt-config_0.8.13-1_all.deb
sudo dpkg -i mysql-apt-config_0.8.13-1_all.deb
```

2. MySQL-Server 설치

```
sudo apt-get update
sudo apt-get install mysql-server
```

3. MySQL 외부 원격 접속 설정

- /etc/mysql/mysql.conf.d/mysqld.cnf 파일 수정
- bind-address 127.0.0.1을 0.0.0.0으로 수정

4. MySQL 접속 계정

4-1. 접속 및 버전 확인

```
sudo /usr/bin/mysql -u root -p
# 아직 비밀번호 세팅이 안되었기때문에 그냥 엔터

show variables like "%version%";
```

4-2. root 계정 비밀번호 변경

```
alter user 'root'@'localhost' identified with mysql_native_password by 'new password';
flush privileges;
```

4-3. 유저 생성 및 권한 설정

```
create user '[username]'@'%' identified by '[password]';
grant all privileges on *.* to '[username]'@'%' with grant option;
flush privileges;
```

Redis 설치

1. 설치

- apt-get update

```
sudo apt-get update
sudo apt-get upgrade
```

- redis 설치 및 버전 확인

```
sudo apt-get install redis-server
redis-server --version
```

2. 외부 접속 허용 및 비밀번호 설정

- /etc/redis/redis.conf에서 수정

```
# bind 127.0.0.1
bind 0.0.0.0:

# requirepass 주석풀어서
requirepass 비밀번호
```

MongoDB 설치

- 자세한 명령어는 밑의 공식문서 참고

<https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/#install-mongodb-community-edition-using-deb-packages>

1. 우분투 버전 확인(공식문서 보고 맞는 방법으로 설치)

```
lsb_release -dc
```

2. 공개키 가져오기

```
wget -q0 - https://www.mongodb.org/static/pgp/server-6.0.asc | sudo apt-key add -
```

3. 몽고디비에 대한 목록 파일 만들기

```
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/6.0 multiverse" | sudo tee /etc/apt/sources.list.
```

4. 로컬 패키지 데이터베이스 다시 로드

```
sudo apt-get update
```

5. 몽고디비 패키지 설치(최신버전) 특정 버전 설치 시, 방법 다름

```
sudo apt-get install -y mongodb-org
```

6. 몽고디비 시작 && 실행 확인

```
sudo systemctl start mongod
sudo systemctl status mongod
```

7. 외부에서 연결 위해 설정파일 수정

/etc/mongod.conf 열어서 수정

```
# network interfaces
net:
  port: 27017
  bindIp: 0.0.0.0 # 이부분 변경

security:
  authorization: enabled # 이부분 변경해야 Auth 적용
```

설정

1. 몽고디비 관리자 계정 추가

- mongosh로 mongo-cli 실행

```
mongosh
```

- 데이터베이스 관리자로 전환

```
use admin
```

- 계정 생성

```
db.createUser({
  user: '아이디',
  pwd: '비밀번호',
  roles: ["userAdminAnyDatabase", "dbAdminAnyDatabase", "readWriteAnyDatabase"]
})
```

2. 사용자 계정 생성

사용자 계정을 만들 때는 특정 DB로 가서 만들어야함.

```
use rideus

db.createUser(
  {
    user: "test1",          // user 이름
    pwd: "password",
    roles: [
      { role: "readWrite", db: "mytestdb2" }, // mytestdb2 에 대해 readWrite 권한
      { role: "read", db: "test" }           // test db 에 대해 read 권한
    ]
  }
)
```

3. Auth 설정

/etc/mongod.conf 열어서 수정

이 설정을 해줘야 로그인 안하면 DB 못들어감

```
security:
  authorization: enabled # 이부분 변경해야 Auth 적용
```

4. MongoDB Compass 연결

- 연결 url

```
mongodb://[username:password@]host1[:port1],...hostN[:portN]][/[defaultauthdb][?options]]
```

([]는 생략이 가능하다는 의미입니다.)

```
ex)
mongodb://hihi:1234@123.123.123.123:27017/dbname
```

```
계정 ID: hihi
pwd: 1234
MongoDB가 설치된 곳의 IP: 123.123.123.123
포트: 27017
사용 DB 이름: dbname
```



설정파일 변경하면 항상 재시작

```
sudo systemctl restart mongod
```

Nginx & SSL 적용

1. Nginx 설치

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

2. Certbot

- letsencrypt의 형태로 SSL/TLS 인증서를 무료로 제공하는 라이브러리

Certbot 설치 후 인증서 발급

```
sudo add-apt-repository ppa:certbot/certbot

sudo apt-get update # 해당 저장소에 담긴 패키지 정보를 확인할 수 있도록 업데이트

sudo apt-get install python3-certbot-nginx # certbot 설치

# 설치된 certbot을 이용하여 도메인(example.com)에 대한 SSL 인증서 발급
sudo certbot certonly --nginx -d j7a603.p.ssafy.io

# 다음 경로에 5개의 파일(4개의 .pem, 1개의 readme) 생성 확인
sudo ls -al /etc/letsencrypt/live/j7a603.p.ssafy.io

# 90일마다 만료되는 인증서 자동 갱신
sudo certbot renew --dry-run
```

3. Nginx 설정 파일 수정

- /etc/nginx/sites-available 밑에 test.conf 설정파일 만들기

```
server {
    listen 80; #80포트로 받을 때
    server_name k7a106.p.ssafy.io www.k7a106.p.ssafy.io; #도메인주소, 없을경우 localhost
    return 301 https://j7a603.p.ssafy.io$request_uri;
}

server {
    listen 443 ssl;
    server_name k7a106.p.ssafy.io www.k7a106.p.ssafy.io;

    # ssl 인증서 적용하기
    ssl_certificate /etc/letsencrypt/live/k7a106.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/k7a106.p.ssafy.io/privkey.pem;

    location / { # 프론트엔드
        proxy_pass http://localhost:3000;
    }

    location /api { # 백엔드
        proxy_pass http://localhost:8080;
        proxy_set_header Host $http_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-Proto $scheme; # https 필요

        # 웹 소켓 설정
        proxy_set_header Connection "upgrade";
        proxy_set_header Upgrade $http_upgrade;

        # SSE 때문에 적용
        # buffer기능을 통해 데이터가 어느정도 쌓여야 보내지는데 SSE 데이터가 너무 작을
        # 메시지가 안보내지는 현상 발생. buffer기능을 꺼서 바로 데이터가 보내지게 설정.
        # (서버에 안충을 수 있음.)
        proxy_buffering off;

        # Nginx에서 WAS 요청시 HTTP/1.0 통신 -> SSE 지속 연결 위해 HTTP/1.1로 변경(HTTP 1.1에서는 기본적으로 지속 연결 사용)
        proxy_set_header Connection '';
        proxy_http_version 1.1;
    }
}
```

- sites-enabled에 심볼릭 링크 설정

```
sudo ln -s /etc/nginx/sites-available/test.conf /etc/nginx/sites-enabled
```

- /etc/nginx/nginx.conf의 http block에 해당 구문 설정(요청 데이터 사이즈 설정)

```
http {
    client_max_body_size 10M;
    ...
}
```

- nginx 재시작

```
sudo service nginx restart # nginx 재시작
```

Jenkins

설치

sudo 없이 docker 명령어 사용하기

sudo 없이 docker 명령어를 사용하려면, docker 그룹에 사용자를 추가해야 함

```
sudo usermod -aG docker $USER
```

```
# 로그아웃 후 재접속  
exit
```

1. Docker에 Jenkins 설치 후 실행(9090포트로 실행)

```
sudo docker run -d \  
-u root \  
-p 9090:8080 \  
--name=jenkins \  
-v /home/ubuntu/docker/jenkins-data:/var/jenkins_home \  
-v /var/run/docker.sock:/var/run/docker.sock \  
-v "$HOME":/home/jenkinsci/blueocean \  
jenkinsci/blueocean
```

- v 옵션이 붙어있습니다. 무엇일까요?

사실 위 옵션 없이 하나의 인스턴스로 도커 파일을 배포하려면, 젠킨스도 컨테이너이기 때문에 젠킨스 안에서 컨테이너를 실행해야 합니다. 하지만 젠킨스 컨테이너 안에서 또 도커를 설치하여 Java안에서 또 컨테이너를 동작시킨다면, 성능이 심각하게 많이 떨어질 것입니다.

이를 해결하기 위해서 docker.sock 파일을 마운트한채로 컨테이너를 동작시켜줘야 합니다. 이렇게 하면 젠킨스 컨테이너 안의 도커가 바깥의 EC2 인스턴스의 도커와 연결되어 젠킨스 컨테이너 안에서 외부 도커에 명령을 내릴 수 있게됩니다.

2. Docker container에 접속

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

3. admin password 확인

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

4. jenkins에 접속 후 기본 플러그인 설치

- <http://<ec2 도메인주소>:9090> 으로 접속
ex) <http://k7a106.p.ssafy.io/9090>

<https://rainbound.tistory.com/entry/JENKINS-GITLAB-%EC%97%B0%EA%B2%B0>

- Dashboard → Jenkins 관리 → 플러그인 관리에서 Docker, Git, Gitlab, pipeline 등 관련 플러그인 설치

파이프라인 생성 및 설정, Gitlab 연결

1. Gitlab에서 API Token 생성하기

Search page

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

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

 YYYY-MM-DD 

Select a role

 Maintainer 

Select scopes

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

- ☐ **api**
Grants complete read/write access to the API, including all groups and projects, the container registry, and the package registry.
- ☐ **read_api**
Grants read access to the API, including all groups and projects, the container registry, and the package registry.
- ☐ **read_repository**
Grants read-only access to repositories on private projects using Git-over-HTTP or the Repository Files API.
- ☐ **write_repository**
Grants read-write access to repositories on private projects using Git-over-HTTP (not using the API).

- Gitlab의 프로젝트 선택 → 설정 → 액세스 토큰
- Token name: 토큰 이름
Expiration date: 토큰 사용기한. 안적으면 무제한
Select a role: Maintainer로 하는게 좋을듯
Select scopes: 권한부여. 다 체크
- Create project access token 버튼 누르면 토큰 생성 완료.
토큰 복사해서 따로 적어놔야 함.

2. Jenkins에서 Gitlab 연결

- Dashboard → Jenkins 관리 → Configure System
Gitlab탭에서 설정

Gitlab

☒ Enable authentication for '/project' end-point

GitLab connections

Connection name

A name for the connection

gitlab_rideus

Gitlab host URL

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

https://lab.ssafy.com

Credentials

API Token for accessing Gitlab

GitLab API token

+ Add

고급...

Test Connection

추가

3. Pipeline 생성 및 설정

- Dashboard → 새로운 Item → pipeline

General

Enabled ☒

설명

[Plain text] [미리보기](#)

- ☐ Do not allow concurrent builds
- ☐ Do not allow the pipeline to resume if the controller restarts
- ☐ GitHub project

GitLab Connection

gitlab_rideus

- ☐ Use alternative credential
- ☐ Pipeline speed/durability override ?
- ☐ Preserve stashes from completed builds ?
- ☐ Throttle builds ?
- ☐ 오래된 빌드 삭제 ?
- ☐ 이 빌드는 매개변수가 있습니다 ?

- 위에서 만든 GitLab Connection 연결해주기

Build Triggers

☐ Build after other projects are built ?

☐ Build periodically ?

☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://j7a603.p.ssfy.io:9090/project/rideus-jenkins> ?

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...

☐ GitHub hook trigger for GITScm polling ?

☐ Poll SCM ?

☐ Quiet period ?

☐ 빌드를 원격으로 유발 (예: 스크립트 사용) ?

- Build when a change is pushed to GibLab 체크박스 선택 후 URL 복사
- 아래에 고급 버튼 누른 후 secret token 생성 후 복사

Gitlab으로 이동

- Gitlab으로 이동 후 설정 → Webhooks

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.

☐ 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.

☐ Job events
A job's status changes.

☐ Pipeline events
A pipeline's status changes.

☐ Wiki page events
A wiki page is created or updated.

☐ Deployment events
A deployment starts, finishes, fails, or is canceled.

☐ Feature flag events
A feature flag is turned on or off.

☐ Releases events
A release is created or updated.

SSL verification

☒ Enable SSL verification

Add webhook

- URL과 Secret token에 복사한 값 넣어주기 → Trigger 원하는거 체크 → Add webhook
- 생성된 Webhook이 아래에 있음. Test → Push events → HTTP:200 뜨면 Webhook 설정 완료

다시 Jenkins로 돌아와서

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://lab.ssfy.com/s07-bigdata-dist-sub2/s07P22A603

Credentials ?

rjeowornjs@naver.com/*****

+ Add

고급...

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

*/back

Add Branch

Repository browser ?

(자동)

Additional Behaviours

Add -

Script Path ?

Jenkinsfile

☒ Lightweight checkout ?

저장 Apply

- 설정 방법 pipeline script와 SCM 두가지 방법이 있음. 여기선 SCM 사용
 - pipeline script: jenkins내에서 스크립트를 입력하는 방법. 젠킨스 내에서 관리
 - SCM: 외부에서 스크립트로 관리할 수 있음.
- Repository URL: Gitlab 프로젝트 주소
 Credentials: Jenkins 설정 → Credentials manage → kind 설정을 username password로 하고 gibrat 계정, 비밀번호 입력
 Branch Specifier: 빌드할 브랜치
 Script Path: 이용할 스크립트의 위치와 파일명
- 저장 누르면 설정 완료.

<https://rainbound.tistory.com/entry/JENKINS-GITLAB-연결> 참고

Dockerfile 및 Jenkinsfile

프론트엔드 Dockerfile

```
# Dockerfile

FROM node:16.16.0 as builder
# root 에 app 폴더를 생성

WORKDIR /app

COPY . /app

RUN npm install

RUN npm run build
```

```
# work dir 에 build 폴더 생성 /app/build

# host pc의 현재경로의 build 폴더를 workdir 의 build 폴더로 복사
# nginx 이미지를 사용합니다. 뒤에 tag가 없으면 latest 를 사용합니다.
FROM nginx

# nginx 의 default.conf 를 삭제
RUN rm /etc/nginx/conf.d/default.conf

# host pc 의 nginx.conf 를 아래 경로에 복사
COPY ./nginx.conf /etc/nginx/conf.d

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

# 80 포트 오픈
EXPOSE 80

# container 실행 시 자동으로 실행할 command. nginx 시작함
CMD ["nginx", "-g", "daemon off;"]
```

프론트엔드 Jenkinsfile

```
pipeline {
    agent any
    environment {
        FRONT_CONTAINER_NAME="weplog_front_container"
        FRONT_NAME = "weplog_front"
    }
    stages {
        stage('Clean'){
            steps{
                script {
                    try{
                        sh "docker stop ${FRONT_CONTAINER_NAME}"
                        sleep 1
                        sh "docker rm ${FRONT_CONTAINER_NAME}"
                    }catch(e){
                        sh 'exit 0'
                    }
                }
            }
        }
        stage('Build') {
            steps {
                script{
                    sh "docker build -t ${FRONT_NAME} ./frontend/"
                }
            }
        }
        stage('Docker run') {
            steps {
                sh "docker run -d --name=${FRONT_CONTAINER_NAME} -p 3000:80 ${FRONT_NAME}"
                sh "docker image prune --force" #기존 도커 이미지 삭제
            }
        }
    }
}
```

백엔드 Dockerfile

```
# 각 서비스마다 이름만 다르게 지정하면 됨
FROM openjdk:8
ADD ./build/libs/member-service-0.0.1-SNAPSHOT.jar MemberService.jar
CMD ["java", "-jar", "ApigatewayService.jar"]
```

```
# Dockerfile 작성 후 image build 및 Docker hub에 push
docker build -t wornjs135/beedly_spring .
docker push wornjs135/beedly_spring
```

백엔드 설정파일

weplog.yml

```

spring:
  # redis
  cache:
    type: redis
  redis:
    host: k7a1061.p.ssafy.io
    port: 6379
    password: <redis 비밀번호>

  # 파일 업로드
  servlet:
    multipart:
      enabled: true
      max-file-size: 10MB
      max-request-size: 100MB

  # swagger 설정
  mvc:
    pathmatch:
      matching-strategy: ant_path_matcher

  # DB 설정
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://<도메인 주소 or ip 주소>:3306/weplog?useSSL=false&useUnicode=true&autoReconnect=true&characterEncoding=utf8&allowMu
    username: <DB 계정>
    password: <비밀번호>

  # JPA 설정
  jpa:
    # database-platform: org.hibernate.dialect.MySQL5InnoDBDialect
    database-platform: org.hibernate.spatial.dialect.mysql.MySQL56InnoDBSpatialDialect
    hibernate:
      # ddl-auto: update
    properties:
      hibernate:
        default_batch_fetch_size: 1
        # show_sql: false
        format_sql: true
        show_sql: true

    open-in-view: false
  web:
    pageable:
      default-per-size: 20
      max-page-size: 2000

  # MongoDB
  data:
    mongodb:
      auto-index-creation:
        uri: mongodb://<DB 계정>:<비밀번호>@<도메인 주소 or ip 주소>:27017/<사용 DB이름>
      # host: <도메인 주소 or ip 주소>
      # port: 27017
      # database: <사용 DB이름>

  # 배치 스케줄링
  task:
    scheduling:
      pool:
        size: 10

  # zipkin
  zipkin:
    base-url: http://127.0.0.1:9411
    enabled: true

  # sleuth
  sleuth:
    sampler:
      probability: 1.0

  # rabbitmq
  rabbitmq:
    host: 127.0.0.1
    port: 5672
    username: guest
    password: guest

  # 게이트웨이
  gateway:
    ip: 127.0.0.1

  # S3
  cloud:

```

```

aws:
  credentials:
    accessKey: <AWS IAM Access Key>
    secretKey: <AWS IAM Secret Key>
  s3:
    bucket: <S3 버킷 이름>
  region:
    static: ap-northeast-2
  stack:
    auto: false

# actuator endpoint
management:
  endpoints:
    web:
      exposure:
        include: refresh, health, beans, busrefresh, info, metrics, prometheus

```

application.yml

```

server:
  port: 8082

spring:
  application:
    name: <서비스 이름>

# rabbitmq
rabbitmq:
  host: 127.0.0.1
  port: 5672
  username: guest
  password: guest

# eureka discovery 등록 정보
eureka:
  instance:
    instance-id: ${spring.application.name}:${spring.application.instance_id:${random.value}}
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://127.0.0.1:8761/eureka

logging:
  level:
    com.ssafy.challengeservice: DEBUG

kafka-ip: 127.0.0.1:9092

```