



# 포팅 메뉴얼

## 버전 설명

### 도커환경

- EC2 도커 버전 : 27.2.1
- 젠킨스 컨테이너 : 2.452.3 (latest)
- Mysql 컨테이너 : 9.0.0-1.el9 (latest)
- nginx 컨테이너 : 1.27.0 (latest)
- Python 컨테이너 : continuumio/miniconda3
- 스프링 부트 컨테이너 : openjdk:17-alpine
- 프론트 컨테이너 : node:20-alpine, nginx:stable-alpine

## 실행방법

### ▼ EC2에 도커 설치

1. `sudo apt update` 로 패키지 업데이트 반영
  - a. 만약 update가 안된다면 bashtop과 certbot문제일 가능성이 높다.
  - b. 업데이트가 불가능한 특정 레포지토리가 bashtop, certbot이라면 아래 명령어를 쓰면 된다.

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

```
sudo apt-add-repository -r ppa:bashtop-monitor/bashtop
```

2.

### apt가 https 저장소를 사용할 수 있게 해주는 Package들을 설치

- `sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common`

## 도커의 공식 GPG 키 추가, Docker 저장소 키를 apt에 등록하기

- `sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -`

## Docker 다운로드 및 repository 리스트에 추가

- `sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"`
- `sudo apt update` 해주기

## docker-ce 설치

| docker-ce는 docker의 무료로 제공되는 docker engine임

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

## docker-compose 설치

- `sudo apt install docker-compose`

## ▼ 젠킨스 설치

### 젠킨스 이미지 다운로드

- `sudo docker pull jenkins`

### 젠킨스 컨테이너 띄우기

- `sudo docker run -d -p 8082:8080 -v /home/ubuntu/jenkins-data:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock --name jenkins -u root jenkins/jenkins:lts`
  - **-d** : 컨테이너를 **데몬**으로 띄웁니다.(백그라운드)
  - **-p 8080:8080** : 컨테이너 외부와 내부 포트를 **포워딩**합니다. **좌측이 호스트 포트, 우측이 컨테이너 포트**입니다.
  - **-v /jenkins:/var/jenkins\_home** : 도커 컨테이너의 데이터는 **컨테이너가 종료되면 휘발**됩니다. 도커 컨테이너의 데이터를 보존하기 위한 여러 방법이 존재하는데, 그 중 한 방법이 **볼륨 마운트**입니다. 이 옵션을 사용하여 젠킨스 컨테이너의 `/var/jenkins_home` 이라는 디렉토리를 호스트의 `/jenkins` 와 마운트하고 데이터를 보존할 수 있습니다.
  - **--name jenkins** : 도커 컨테이너의 이름을 설정합니다.
  - **-u root** : 컨테이너가 실행될 리눅스의 사용자 계정을 root 로 명시합니다



만약 컨테이너를 재 실행하고 싶으면 `sudo docker start jenkins`만 하면 됨  
(-d나 -p, -v같은 옵션들은 자동적으로 저장되어서 들어감)

- jenkins 내에서 docker를 실행 시킬수 있게 하기 위해 docker-ce 설치

```
apt-get update && \
apt-get -y install apt-transport-https \
    ca-certificates \
    curl \
    gnupg2 \
    software-properties-common && \
curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo ${ID}) \
add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo ${ID}) \
    $(lsb_release -cs) \
    stable" && \
apt-get update && \
apt-get -y install docker-ce
```

- 추가적으로 docker jenkins에서 host docker 접근 권한 부여

```
groupadd -f docker

usermod -aG docker jenkins

chown root:docker /var/run/docker.sock
```

#### ▼ 해석

```
groupadd -f docker
```

groupadd: 새로운 그룹을 생성하는 명령어입니다.

-f: 이미 그룹이 존재하면 아무 작업도 하지 않고 종료합니다. (그룹이 존재하지 않으면 생성)

docker: 생성할 그룹의 이름입니다. 이 명령어는 docker라는 이름의 그룹을 생성합니다.

```
usermod -aG docker jenkins
```

usermod: 사용자를 수정하는 명령어입니다.

-aG: 사용자를 새로운 그룹에 추가하는 옵션입니다. -a는 추가를 의  
docker: 사용자가 추가될 그룹의 이름입니다.

jenkins: 그룹에 추가할 사용자의 이름입니다. 이 명령어는 jenkins  
chown root:docker /var/run/docker.sock

chown: 파일의 소유자와 그룹을 변경하는 명령어입니다.

root:docker: 소유자를 root로, 그룹을 docker로 설정합니다.

/var/run/docker.sock: 소유자와 그룹을 변경할 대상 파일입니다

- docker-compose를 실행하기 위해 docker-compose 설치

```
curl -L "https://github.com/docker/compose/releases/download/1.25.0/docker-compose" -o /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
```

```
curl -L "https://github.com/docker/compose/releases/download/1.25.0/docker-compose" -o /usr/local/bin/docker-compose
chmod +x /usr/local/bin/docker-compose
```

- 도커파일로 정리하자면?

```
FROM jenkins/jenkins:latest
```

```
ENV DEBIAN_FRONTEND noninteractive
```

```
ENV DEBCONF_NOWARNINGS="yes"
```

```
ARG HOST_UID=502
```

```
USER root
```

```
RUN apt-get -y update && apt-get install -y --no-install-recommends vim \
    apt-utils
```

```
RUN apt-get install ca-certificates curl gnupg lsb-release
```

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

```
RUN curl -fsSL https://download.docker.com/linux/debian/gpg > /etc/apt/keyrings/docker.asc
```

```
RUN echo "deb [arch=$(dpkg --print-architecture) signed-by=
```

```

RUN apt-get -y update
RUN apt-get install docker-ce docker-ce-cli containerd.io
RUN if [ -e /var/run/docker.sock ]; then chown jenkins:jenkins /var/run/docker.sock; fi

RUN usermod -u $HOST_UID jenkins
RUN usermod -aG docker jenkins

USER jenkins

```

## docker-compose 작성

도커를 실행할 경로에 docker-compose.yml 파일을 만들고 아래 내용 작성하기

- 그 후 실행하는 명령어 `sudo docker-compose up -d`

## ▼ 젠킨스 설정

만약 정상적으로 설치 되었다면 브라우저를 통해 외부에서 접속가능

- `i11b110.p.ssafy.io`: 포트번호
- 처음 접속하면 비밀번호를 입력하라고 나온다
- `sudo docker logs jenkins` 로 로그를 확인하면 비밀번호가 나온다
- 아이디: admin
- 비밀번호 : ada510f9f87f47cda9423971ea9ed12c

## 플러그인 설치

- 처음에 권장되는 플러그인을 설치받는다
- 만약 설치가 정상적으로 안된다면 아래 과정을 그대로 해준다

```

cd /jenkins

mkdir update-center-rootCAs

wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-cent

```

```
sudo sed -i 's#https://updates.jenkins.io/update-center.jsc

sudo docker restart jenkins
```

- 그 후 `hudson.model.UpdateCenter.xml` 파일을 열어 링크가 위에 `sudo sed...`처럼 되어있는지 확인함
- 그 후 설정 → Account → User Defined Time Zone 에서 Asia/Seoul 로 설정
- gitlab 플러그인만 따로 설치해주면 젠킨스 설정 완료!

## ▼ 웹훅 설정하기

본격적인 CI/CD설정을 하기 앞서 기존 배포 프로세스를 설명하자면

1. 소스코드를 git clone을 통해 다운로드 받는다.
2. 서버에 Java를 설치한 후 `./gradlew` 명령어로 실행파일인 jar파일을 생성한다.
3. `java -jar` 명령어로 서비스를 실행
4. nginx를 설치, 80포트 바인딩

### 도커파일로 반 자동화



Dockerfile은 로컬환경에서 프로젝트 루트디렉토리에 작성하자

## Dockerfile 명령어

```
FROM openjdk:11-jdk-slim-buster
COPY build/libs/login-service.jar app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

**FROM:** 생성할 이미지의 베이스 이미지

**COPY:** 로컬의 파일을 이미지에 복사

**EXPOSE:** 노출할 포트 지정

**ENTRYPOINT:** 컨테이너가 시작될 때 수행할 명령어를 지정

75


그 후에 ./gradlew clean build로 빌드 폴더 생성 후

`docker build -t [태그명]:[버전명] ./ (도커파일위치)` 로 이미지 생성하기

그 후 위 이미지를 컨테이너에 올리면 도커파일에 명시된 명령어들이 자동 실행된다.

### [Jenkins] Jenkins GitLab 연동하기

1. Jenkins 관리의 ManageCredentials로 이동한다. 2. Add credentials를 선택하여 - kind : Username with password - Scope : Global (Jenkins, nodes, items, all child items,

 <https://ojm1484.tistory.com/58>



## ▼ 도커에 mysql 설치

- `docker pull mysql`
- `sudo docker run --name mysql -e MYSQL_ROOT_PASSWORD='11p12b304supia!@' -e MYSQL_DATABASE=supia -e TZ=Asia/Seoul -v /mysql/data:/var/lib/mysql -v /mysql/config:/etc/mysql/conf.d -v /mysql/init:/docker-entry-point-initdb.d -d -p 3306:3306 --network=supia mysql:latest`

- -v로 config파일과 init파일과 mysqldata들을 호스트 볼륨에 매핑해주기
- `docker exec -it [컨테이너명] bash` 로 해당 컨테이너 터미널 접속
- `mysql -u root -p` 로 mysql 실행
- 비밀번호 입력후 `show databases;` 입력해서 db 확인하기
- 나중에 docker-compose 로 할때 timezone 설정 잊지말기

## ▼ 파이프라인 설정하기 (CI)

일단 파이프라인에서 nodeJS를 설치하자

### Global Tool Configuration에서 Nodejs, Gradle 추가

- 버전은 따로 지정해주면됨
- Gradle도 똑같은 방법으로 추가, 버전은 인텔리제이에 나와있음

### Git 스크립트 구성하기

```
pipeline {
    agent any

    tools {
        nodejs "nodejs"
        gradle "gradle"
    }
}
```



```

    stages {
        stage('git') {
            steps {
                git branch: 'develop', credentialsId: 'mpecc
            }
        }
    }
}

```

- 위 명령어를 실행하면 해당 url을 clone받아온다.
- 클론받은 파일들은 jenkins\_home에 workspace공간에 있음 (당연히 volume으로 연결한 호스트 디렉토리에 존재한다)

## ▼ dev-BE 파이프라인

```

pipeline {
    agent any

    stages {
        stage('clone') {
            steps {
                git branch: 'dev-BE', credentialsId: '
            }
        }
        stage('container down') {
            steps {
                sh 'docker stop back || true'
                sh 'docker rm back || true'
            }
        }
        stage('application.properties download') {
            steps {
                dir('supia-back'){
                    withCredentials([file(credentialsId:
                        script {
                            sh 'cp $applicationPropertie
                            sh 'ls src/main/resources'
                        }
                    )
                }
            }
        }
    }
}

```

```

sh 'cat src/main/resources/a
    }
    }
    }
    }
    }
stage('back_build'){
    steps{
        dir('supia-back'){
            sh 'chmod +x gradlew'
            sh './gradlew clean build'
            sh 'docker rmi -f back:latest'
            sh 'docker build -t back .'
        }
    }
}
stage('deploy'){
    steps{
        sh 'docker run --name back --expose=8080
    }
}
}
}

```

## ▼ dev-FE 파이프라인

```

pipeline {
    agent any

    stages {
        stage('clone') {
            steps {
                git branch: 'dev-FE', credentialsId: '
            }
        }
        stage('container down') {
            steps {
                sh 'docker stop front || true'
                sh 'docker rm front || true'
            }
        }
    }
}

```

```

    }
  }
  stage('front_build'){
    steps{
      dir('supia-front/supia_app'){
        sh 'docker rmi -f front:latest'
        sh 'docker build -t front .'
      }
    }
  }
  stage('deploy'){
    steps{
      sh 'docker run --name front -v /etc/lets
    }
  }
}

```

## ▼ develop 파이프라인

```

pipeline {
  agent any

  stages {
    stage('clone') {
      steps {
        git branch: 'develop', credentialsId: '
      }
    }
    stage('docker-compose down') {
      steps {
        sh 'docker-compose -f /var/jenkins_home/
      }
    }
    stage('front_build'){
      steps{
        dir('supia-front/supia_app'){
          sh 'docker rmi -f front:latest'
          sh 'docker build -t front .'
        }
      }
    }
  }
}

```

```

    }
  }
}
stage('application.properties download') {
  steps {
    dir('supia-back'){
      withCredentials([file(credentialsId:
        script {
          sh 'cp $applicationPropertie
          sh 'ls src/main/resources'
          sh 'cat src/main/resources/a
        }
      }
    }
  }
}
stage('back_build'){
  steps{
    dir('supia-back'){
      sh 'chmod +x gradlew'
      sh './gradlew clean build'
      sh 'docker rmi -f back:latest'
      sh 'docker build -t back .'
    }
  }
}
stage('deploy'){
  steps{
    sh 'docker-compose -f /var/jenkins_home/
  }
}
}
}

```

## ▼ dev-AI 파이프라인

```

pipeline {
  agent any

```

```

stages {
  stage('clone') {
    steps {
      git branch: 'dev-AI', credentialsId: '
    }
  }
  stage('container down') {
    steps {
      sh 'docker stop supia-ai || true'
      sh 'docker rm supia-ai || true'
    }
  }
  stage('.env download') {
    steps {
      withCredentials([file(credentialsId: 'ai
        script {
          sh 'cp $aiEnv .'
          sh 'ls .'
          sh 'cat ../.env'
        }
      ]
    }
  }
  stage('AI_build'){
    steps{
      sh 'docker rmi -f supia-ai:latest'
      sh 'docker build -t supia-ai .'
    }
  }
  stage('deploy'){
    steps{
      sh 'docker run --name supia-ai --expose=
    }
  }
}
}

```

## ▼ 프론트 엔드 도커 파일 설정하기

- 멀티스테이지 빌드를 활용해서 바로 nginx에 html파일을 올린다

```
FROM node:20-alpine as build
WORKDIR /app
# 컨테이너 내에서의 워킹 디렉토리 변경

COPY package*.json ./
#먼저 패키지 인스톨
RUN npm install
COPY . .
# 그 후 소스코드들을 복붙
RUN npx expo export --platform web
#expo에서 웹 빌드하는 명령어

FROM nginx:stable-alpine as production-stage
COPY --from=build /app/dist /usr/share/nginx/html
EXPOSE 5173

CMD ["nginx", "-g", "daemon off;"]
```

- --from 은 첫 빌드 스테이지 내에서 경로 지정

## ▼ 리버스 프록시용 nginx config 설정

```
sudo docker run --name nginx -d -v /home/ubuntu/nginx/templates:/etc/nginx/templates -v /etc/letsencrypt:/etc/letsencrypt -p 80:80 -p 443:443 --network=supia nginx:latest
```

- templates 를 연동하는게 중요함
- letsencrypt는 https 설정을 위해

### ▼ 옛날방법

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

    server_name ssayeon.co.kr;

    large_client_header_buffers 4 32k;
```

```

    root /usr/share/nginx/html;

    location / {
        root /home/ubuntu/Frontend/dist; # 정적 I
        index index.html; # index.html 띄워주기
        try_files $uri $uri/ /index.html;
    }

    location /api {
        proxy_pass http://spring:8081;
        proxy_redirect off;
        charset utf-8;

        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_
        proxy_set_header X-Forwarded-Proto $sche
        proxy_set_header X-NginX-Proxy true;

        client_max_body_size 500M;

        # 502 에러를 없애기 위한...
        proxy_buffer_size          128k;
        proxy_buffers                4 256k;
        proxy_busy_buffers_size     256k;

        proxy_connect_timeout 300s;
        proxy_read_timeout 600s;
        proxy_send_timeout 600s;
    }
}

```

- 옳은 방법은 ec2 templates 폴더에 임의파일명.conf.template을 만들어서 거기다가

```

server {
    listen 443 ssl;
    server_name j11b208.p.ssafy.io;
}

```

```

    access_log /var/log/nginx/access.log;
    access_log /dev/stdout;

    error_log /var/log/nginx/error.log;
    error_log /dev/stderr;
    ssl_certificate /etc/letsencrypt/live/p.ssafy.io;
    ssl_certificate_key /etc/letsencrypt/live/p.ssafy.io/privkey.pem;

    location / {
        proxy_pass http://front:5173;
        add_header 'Cross-Origin-Embedder-Policy' 'cross-use-credentials';
        add_header 'Cross-Origin-Opener-Policy' 'cross-use-credentials';
    }

    location /api{
        charset utf-8;
        proxy_pass http://back:8080;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        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;
        proxy_set_header Host $http_host;

    }

    location /radios{
        proxy_pass http://radio:8081;
        proxy_set_header Host $http_host;
    }

    location /chatAI{
        proxy_pass http://rag:7777;
    }

    location /summary{

```



```

        proxy_pass http://summary:3333;
    }
}

```

- 위 파일 만들어주기
- 그러면 자동으로 include 된다
- 여기서 주의할점은 proxy\_pass에 컨테이너명을 적어줘야 한다는 것
  - 왜냐하면 풀 url을 적으면 ufw가 안뚫려있기 때문에 곤란

## ▼ 프론트 엔드 용 nginx config 설정

```

server {
    listen 5173;
    server_name j11b208.p.ssafy.io;
    location / {

        root /usr/share/nginx/html/;
        index index.html index.htm;

        add_header 'Cross-Origin-Embedder-Policy' '
        add_header 'Cross-Origin-Opener-Policy' 'sa

    }
}

```

## ▼ 시용 도커파일

### 벡터서치 컨테이너

```

FROM nvidia/cuda:11.8.0-cudnn8-runtime-ubuntu22.04

RUN apt-get update && apt-get install -y \
    wget \
    bzip2 \
    ca-certificates \
    curl \
    git \

```

```

    build-essential \
    swig \
    libmecab-dev \
    mecab \
    mecab-ipadic-utf8 \
    && rm -rf /var/lib/apt/lists/*

# MeCab-ko 및 MeCab-ko-dic 설치
RUN cd /tmp && \
    curl -LO https://bitbucket.org/eunjeon/mecab-ko/download \
    tar zxvf mecab-0.996-ko-0.9.2.tar.gz && \
    cd mecab-0.996-ko-0.9.2 && \
    ./configure && \
    make && \
    make check && \
    make install && \
    ldconfig && \
    cd /tmp && \
    curl -LO https://bitbucket.org/eunjeon/mecab-ko-dic/download \
    tar -zxvf mecab-ko-dic-2.1.1-20180720.tar.gz && \
    cd mecab-ko-dic-2.1.1-20180720 && \
    ./configure && \
    make && \
    make install

# Conda 설치
RUN wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -b -p /opt/conda \
    bash Miniconda3-latest-Linux-x86_64.sh -b -p /opt/conda \
    rm Miniconda3-latest-Linux-x86_64.sh

ENV PATH="/opt/conda/bin:$PATH"

# 작업 디렉토리 설정
WORKDIR /app

# 환경 파일을 컨테이너로 복사
COPY environment.yml /app/environment.yml

```

```

# Conda 환경 생성
RUN conda env create -f /app/environment.yml

# pip 업그레이드
RUN conda run -n doc2vec pip install --upgrade pip setuptools

# Conda 환경 활성화 (bash 쉘에서 작동하도록 설정)
SHELL ["conda", "run", "-n", "doc2vec", "/bin/bash", "-c"]

# 애플리케이션 코드 복사
COPY . /app

# MeCab 설정 파일 복사 (필요한 경우)
RUN mkdir -p /usr/local/etc && \
    echo "dicdir=/usr/local/lib/mecab/dic/mecab-ko-dic" > /

# 포트 노출
EXPOSE 8000

# 컨테이너 시작 시 FastAPI 서버 실행
ENTRYPOINT ["conda", "run", "--no-capture-output", "-n", "c

# # Miniconda가 포함된 기본 이미지 사용
# # FROM continuumio/miniconda3
# FROM nvidia/cuda:11.8.0-cudnn8-runtime-ubuntu22.04

# RUN apt-get update && apt-get install -y \
#     wget \
#     bzip2 \
#     ca-certificates \
#     curl \
#     git \
#     build-essential \
#     swig \
#     libmecab-dev \
#     mecab \
#     mecab-ipadic-utf8 \
#     && rm -rf /var/lib/apt/lists/*

```

```

# # Conda 설치
# RUN apt-get update && apt-get install -y wget && \
#     wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -b -p /opt/conda/
#     bash Miniconda3-latest-Linux-x86_64.sh -b -p /opt/conda/
#     rm Miniconda3-latest-Linux-x86_64.sh

# ENV PATH="/opt/conda/bin:$PATH"

# # 작업 디렉토리 설정
# WORKDIR /app

# # 환경 파일을 컨테이너로 복사
# COPY environment.yml /app/environment.yml

# # Conda 환경 생성
# RUN conda env create -f /app/environment.yml

# # pip 업그레이드
# RUN conda run -n doc2vec pip install --upgrade pip setuptools

# # Conda 환경 활성화 (bash 쉘에서 작동하도록 설정)
# SHELL ["conda", "run", "-n", "doc2vec", "/bin/bash", "-c"]

# # 애플리케이션 코드 복사
# COPY . /app

# # 포트 노출
# EXPOSE 8000

# # 컨테이너 시작 시 FastAPI 서버 실행
# ENTRYPOINT ["conda", "run", "--no-capture-output", "-n", "doc2vec", "python", "main.py"]

```

## 요약 컨테이너

```

# Use Miniconda as a base image
FROM continuumio/miniconda3

```

```

# Set environment variables
ENV LANG=C.UTF-8 \
    LC_ALL=C.UTF-8

# Set the working directory in the container
WORKDIR /app_overview

# Copy the environment specification to the container
COPY environment.yml /app_overview

# Create the environment and activate it
RUN conda config --add channels defaults && \
    conda config --add channels conda-forge && \
    conda env create -f environment.yml

# Add Conda to the PATH environment variable
ENV PATH /opt/conda/envs/paper-rag/bin:$PATH

# Install necessary system packages
RUN apt-get update && apt-get install -y \
    wget \
    unzip \
    libnss3 \
    libgconf-2-4 \
    libxss1 \
    libappindicator3-1 \
    libasound2 \
    fonts-liberation \
    xdg-utils \
    libgtk-3-0 \
    libgbm-dev \
    libgl1-mesa-glx \
    libxcomposite1

# Copy the application code to the container
COPY . /app_overview

# Chrome 다운로드

```

```

RUN wget https://storage.googleapis.com/chrome-for-testing-
RUN wget https://storage.googleapis.com/chrome-for-testing-

RUN unzip chrome-linux64.zip
RUN unzip chromedriver-linux64.zip

# Grant execution permission to Chromedriver and Chrome bin
RUN chmod +x /app_overview/chromedriver-linux64/chromedriver
RUN chmod +x /app_overview/chrome-linux64/chrome

# Expose the port that the FastAPI application will run on
EXPOSE 3333

# Define the command to run the application
CMD ["bash", "-c", "source activate paper-overview && uvicorn"]

```

## RAG 컨테이너

```

# Use Miniconda as a base image
FROM continuumio/miniconda3

# Set environment variables
ENV LANG=C.UTF-8 \
    LC_ALL=C.UTF-8

# Set the working directory in the container
WORKDIR /app_rag

# Copy the environment specification to the container
COPY environment.yml /app_rag

# Create the environment and activate it
RUN conda config --add channels defaults && \
    conda config --add channels conda-forge && \
    conda env create -f environment.yml

# Add Conda to the PATH environment variable
ENV PATH /opt/conda/envs/paper-rag/bin:$PATH

```

```

# Install necessary system packages
RUN apt-get update && apt-get install -y \
    wget \
    unzip \
    libnss3 \
    libgconf-2-4 \
    libxss1 \
    libappindicator3-1 \
    libasound2 \
    fonts-liberation \
    xdg-utils \
    libgtk-3-0 \
    libgbm-dev \
    libgl1-mesa-glx \
    libxcomposite1

# Copy the application code to the container
COPY . /app_rag

# Chrome 다운로드
RUN wget https://storage.googleapis.com/chrome-for-testing-
RUN wget https://storage.googleapis.com/chrome-for-testing-

RUN unzip chrome-linux64.zip
RUN unzip chromedriver-linux64.zip

# Grant execution permission to Chromedriver and Chrome bin
RUN chmod +x /app_rag/chromedriver-linux64/chromedriver
RUN chmod +x /app_rag/chrome-linux64/chrome

# RUN pip install pymupdf4llm==0.0.17
# RUN pip install pdfplumber==0.11.4

# Expose the port that the FastAPI application will run on
EXPOSE 7777

```

```
# Define the command to run the application
CMD ["bash", "-c", "source activate paper-rag && uvicorn co
```

## ▼ https SSL 설정

```
sudo apt install certbot
sudo certbot certonly --standalone -d i11b304.p.ssafy.io //
sudo certbot certificates
```

## ▼ 백엔드 도커파일 설정하기

```
FROM openjdk:17-alpine
ARG JAR_FILE=build/libs/supia-0.0.1-SNAPSHOT.jar
COPY ${JAR_FILE} /app.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

- 특정포트가 열려있는지 확인하려면 docker ps 가 아니라 해당 컨테이너에 직접 들어가서 `netstat -an` 으로 listen되고있는지 확인해보기

```
# open jdk 21 버전의 환경을 구성
FROM openjdk:21

# tzdata 패키지 설치 및 타임존 설정
RUN ln -snf /usr/share/zoneinfo/Asia/Seoul /etc/localtime &

# build가 되는 시점에 JAR_FILE이라는 변수 명에 build/libs/*.jar
# build/libs - gradle로 빌드했을 때 jar 파일이 생성되는 경로
ARG JAR_FILE=build/libs/ourClass-0.0.1-SNAPSHOT.jar

# JAR_FILE을 agaproject.jar로 복사
COPY ${JAR_FILE} ourClass.jar

# 운영 및 개발에서 사용되는 환경 설정을 분리
# -Duser.timezone=Asia/Seoul JVM 옵션을 사용하여 애플리케이션 수준
ENTRYPOINT ["java", "-jar", "-Dspring.profiles.active=dev",
```



## ▼ 프론트용 도커파일 설정

```
FROM node:16-alpine3.16 AS build
WORKDIR /app

COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
RUN ls ./build

FROM nginx:stable-alpine AS production-stage
COPY --from=build /app/build /usr/share/nginx/html/
EXPOSE 3000

CMD ["nginx", "-g", "daemon off;"]
```

## ▼ 도커 컴포즈 설정 전 네트워크 하나 만들기

`docker network create my-shared-network` 로 네트워크 하나 만들고

도커 컴포즈 파일 내에서

```
networks:
  my-shared-network:
    external: true
```

로 지정해주면 된다.

## ▼ 프론트 도커 컴포즈 설정하기

## ▼ 백엔드 도커 컴포즈 설정하기

```
version: "3"
services:
  nginx:
    build:
      context: ./frontend
    ports:
```

```

    - 80:80
  depends_on:
    - spring

  spring:
    build:
      context: ./backend
    ports:
      - 8082:8081 #로컬포트:컨테이너포트

```

## ▼ Develop 도커 컴포즈 설정하기

```

version: "3"
services:
  front:
    container_name: front
    image: front
    networks:
      - supia
    volumes:
      - /etc/letsencrypt:/etc/letsencrypt
      - /home/ubuntu/front/templates:/etc/nginx/templates

  spring:
    container_name: back
    image: back
    networks:
      - supia

networks:
  supia:
    external: true

```

## ▼ AI 도커 컴포즈 설정하기

## ▼ 기타 도커 컴포즈 설정하기

```

version: "3"
services:
  mysql:
    image: mysql //mysql이 배포한 기본 이미지명
    container_name: mysql //내가 이미지를 돌릴 컨테이너 이름. 아무
    environment: //설정 파일
      MYSQL_DATABASE: supia
      MYSQL_ROOT_PASSWORD:
      TZ: Asia/Seoul
    ports:
      - 3306:3306 //앞: 서버 포트, 뒤: 도커 포트 연결.
    networks:
      - backend

  redis:
    container_name: redis
    image: redis
    ports:
      - 6379:6379
    networks:
      - backend

  react:
    container_name: react
    image: jjongbbang2/talktalk-friu
    expose: //컨테이너 내 이미지들이 연결할 공개 포트
      - 3000
    ports:
      - 3000:3000
    networks:
      - backend

  nginx: //L7 레이어, 프록시 서버 (80 포트)
    container_name: nginx
    image: nginx
    ports:
      - 80:80
      - 443:443

```

```

volumes: //운영서버 경로 : 도커 서버 경로 내 매핑, 지우면 안되는
  - ./nginx/conf.d:/etc/nginx/conf.d
  - /etc/letsencrypt:/etc/letsencrypt
  - /home/ubuntu/nginx/templates:/etc/nginx/templates
restart: always
networks:
  - backend

certbot:
  container_name: certbot
  image: certbot/certbot
  restart: unless-stopped
  volumes:
    - ./data/certbot/conf:/etc/letsencrypt
    - ./data/certbot/www:/var/www/certbot
  entrypoint: "/bin/sh -c 'trap exit TERM; while :; do ce

networks:
  backend:
    driver: bridge

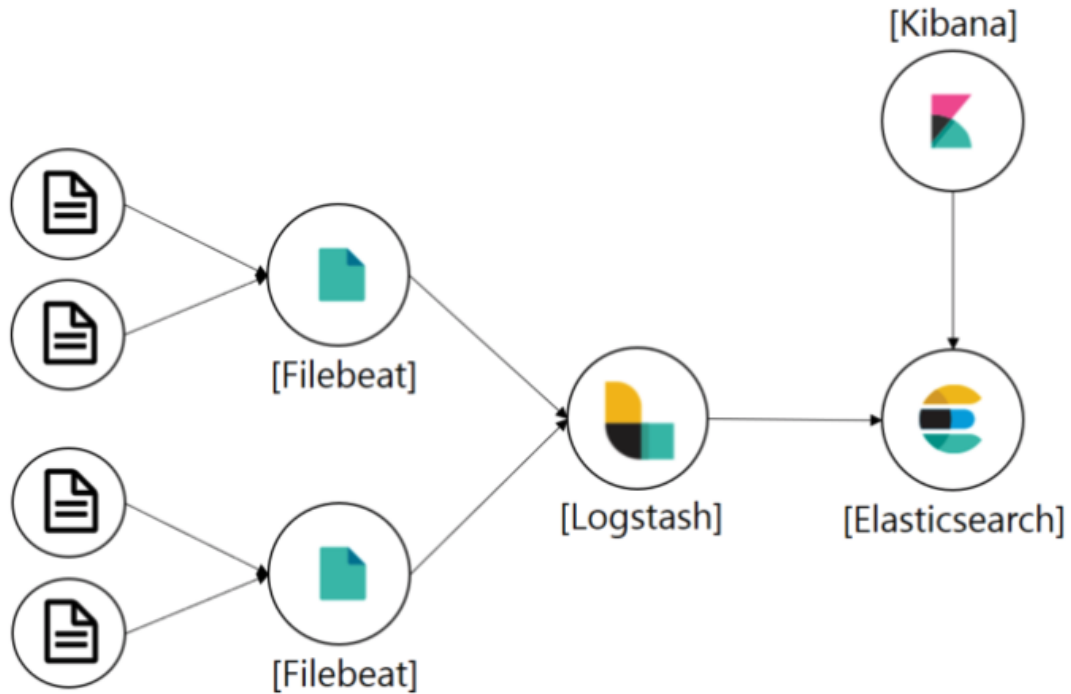
```

## ▼ ELK 스택으로 로그 수집

### 로그를 수집하는 이유

과거처럼 Monolithic 하게 돌아가는 방식이 아니라 MSA로 여러 컨테이너로 쪼개서 서비스를 제공하기 때문에 오류가 생겼을 때 모든 컨테이너에 하나씩 다 들어가서 로그를 보는 방식은 상당히 비효율적이다. 그러므로 각 컨테이너에서 나오는 로그들을 수집해서 한 곳으로 보내면 거기서만 로그를 보면 되니까 매우 간편해짐!!

### 전체적인 구조



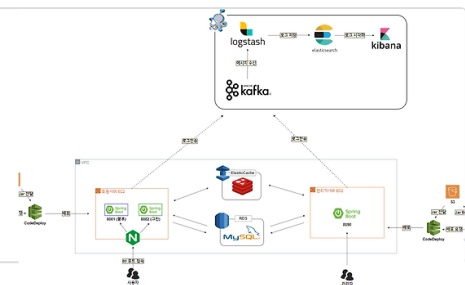
1. logFile 들을 읽어서 Logstash 에 전송
2. Logstash는 수집한 로그들을 filtering 해서 Elasticsearch에 보내줌
3. Kibana는 Elasticsearch에 있는 데이터들을 시각화 해준다.

## Kafka + Kibana 컨테이너 올리기

[Spring] ELK + Kafka를 활용해 실시간 로그 수집하기

ELK 도입 이유ELK가 무엇인지에 대해 알아보고 싶다면 아래 포스팅을 참고. [ELK] ElasticSearch란? ELK란? 내부 구조, 장단점, RDB와 차이ElasticSearch란? Elasticsearch는 Apache

<https://hstory0208.tistory.com/entry/Spring-ELK-Kafka를-활용해-실시간-로그-수집하기>



```
services:
  zookeeper:
    image: wurstmeister/zookeeper:latest
    container_name: zookeeper
    ports:
      - "2181:2181"
    networks:
      - elkstack_elk
  kafka:
    image: wurstmeister/kafka:latest
    container_name: kafka
    ports:
      - "9092:9092"
    networks:
      - elkstack_elk
```

```

image: wurstmeister/kafka:latest
container_name: kafka
depends_on:
  - zookeeper
ports:
  - "9092:9092"
environment:
  KAFKA_BROKER_ID: 1
  KAFKA_ADVERTISED_HOST_NAME: j11b208a.p.ssafy.io
  BOOTSTRAP_SERVERS: j11b208a.p.ssafy.io:9092
  KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
  KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
  KAFKA_CREATE_TOPICS: "back:1:1"
volumes:
  - /var/run/docker.sock:/var/run/docker.sock
networks:
  - elkstack_elk

kafka-ui:
  image: provectuslabs/kafka-ui
  container_name: kafka-ui
  ports:
    - "10000:8080"
  restart: always
  environment:
    - KAFKA_CLUSTERS_0_NAME=local
    - KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS=kafka:9092
    - KAFKA_CLUSTERS_0_ZOOKEEPER=zookeeper:2181
  networks :
    - elkstack_elk

networks:
  elkstack_elk:
    external: true

```

## 카프카 설정

## logstash 설정

```
input {
  kafka {
    bootstrap_servers => "kafka:9092"
    topics => ["back"]
    consumer_threads => 1
    decorate_events => true
  }
}

## Add your filters / logstash plugins configuration here

output {
  elasticsearch {
    hosts => "elkstack-elasticsearch-1:9200"
    user => "elastic"
    password => "CfTFCd1mTt1lIZs=rfkv"
    index => "back-%{+YYYY.MM.dd}"
  }
}
```

## 로그 찍기

로그를 수집하려면 먼저 로그를 생성해야 한다.

현재 수집할 로그들은 Spring Boot와 Python, nginx이다.

- **Spring에서 로그찍기**

- 의존성 추가

```
// Kafka에 로그를 전송하기 위해 Logback에 Kafka Appender를
implementation ("com.github.danielwegener:logback
// Logstash에서 처리할 수 있는 JSON 형식으로 로그 메시지를 인
implementation ("net.logstash.logback:logstash-lc
```

- logback.xml 파일을 resources에 추가

```
<configuration>
  <!-- <appender> KafkaAppender 클래스를 사용하여 Kaf
  <appender name="LOG-KAFKA" class="com.github.dani
    <!-- <encoder> 로그 메시지를 Kafka가 처리할 수 있
    <encoder class="com.github.danielwegener.logb
      <!-- <layout> 로그 메시지 출력 패턴 지정 -->
      <layout class="ch.qos.logback.classic.Pat
        <pattern>%date - %-5p %t %-25logger{5
      </layout>
    </encoder>
    <!-- <topic> Kafka에 전송될 토픽의 이름을 지정 --
    <topic>back</topic>
    <!-- <keyingStrategy> 키 생성 전략을 Round Robin
    <keyingStrategy class="com.github.danielweger
    <!-- <deliveryStrategy> 비동기적 방식으로 로그 메
    <deliveryStrategy class="com.github.danielweg
    <!-- <producerConfig> Kafka producer의 설정값 (
    <producerConfig>retries=1</producerConfig>
    <producerConfig>bootstrap.servers=j11b208a.p.
    <producerConfig>compression.type=snappy</proc
    <producerConfig>max.block.ms=1000</producerCc
  </appender>

  <!-- <appender> 콘솔에 로그를 출력하기 위한 appender
  <appender name="STDOUT" class="ch.qos.logback.cor
    <encoder>
      <pattern>%date - %-5p %t %-25logger{5} %F
    </encoder>
  </appender>

  <!--
  <logger> elk 이름의 logger를 정의하여, INFO 레벨 메시지
  <appender-ref> elk 이름의 logger가 사용할 appender 자
  -->
  <!--
  <logger name="elk" level="INFO" additivity="false
```



```

        <appender-ref ref="LOG-KAFKA"/>
        <appender-ref ref="STDOUT"/>
    </logger>
-->

<!-- <root> 모든 logger에 대한 기본 설정을 정의 (INFO)
<root level="INFO">
    <appender-ref ref="STDOUT"/>
    <appender-ref ref="LOG-KAFKA"/>
</root>
</configuration>

```

○ 클래스에서의 사용

- `private final Logger logger = LoggerFactory.getLogger(this.getClass());`
- @Autowired 같은 의존성 주입은 안해줘도 된다.

▼ 자세한 사용법

```

// 틀린 예)
log.trace("trace log=" + name);

// 옳은 예)
log.trace("trace log={}", name);

```

**첫번째**와 같은 방식으로 trace를 출력할 경우 로그 출력 자체에는 문제가 없다.

하지만 자바 문법상 "trace log="+ name이라는 문자열 더하기 연산이 먼저 일어난 후 log레벨을 확인하기 때문에 만약 해당 프로젝트의 log레벨 설정이 **debug**일 경우 **trace는 debug이하**의 log레벨이므로 출력은 하지 않지만 이미 불필요한 연산 자체가 일어난 경우가 되어 버린다.

**두번째** 방식은 SLF4J 치환문자를 사용하는 방식이다.

문자열에 중괄호를 넣어 순서대로 출력하고자 하는 데이터들을 ' , '로 구분한 후 전달하여 치환해주는 방식으로 불필요한 연산이 발생하지 않도록 한다.