포팅 매뉴얼

버전

- EC2 도커 버전 : `27.3.1`
- 젠킨스 컨테이너 : `2.483`
- mysql 컨테이너 : `9.1.0-1.el9`
- nginx 컨테이너 : `1.27.2`
- 스프링 부트 컨테이너 : `openjdk-17`
- 프론트 nginx 컨테이너 : `1.26.2`
- mosquitto 컨테이너 : `2.0.20`
- elk 컨테이너 : `8.15.3`
- filebeat 컨테이너 : `8.15.3`
- redis 컨테이너 : `7.4.1`

환경 변수

- back

```
spring.application.name=windoorman
server.servlet.context-path=/api

# DataBase Settings : hikariCP : https://github.com/brettwooldridge
spring.datasource.hikari.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://k11b107.p.ssafy.io:3306/windoorm
spring.datasource.hikari.username=root
spring.datasource.hikari.password= ${password}
spring.cache.type=redis
spring.data.redis.host=k11b107.p.ssafy.io
spring.data.redis.port=6379

spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

```
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.dialect= org.hibernate.dialect.MySQ

spring.mustache.prefix=classpath:/templates/
spring.mustache.suffix=.mustache

spring.mvc.view.prefix=/WEB-INF/views/
spring.mvc.view.suffix=.mustache

spring.elasticsearch.uris=http://k11b107.p.ssafy.io:9200
spring.elasticsearch.username=elastic
spring.elasticsearch.password= ${password}

spring.elasticsearch.ssl.certificate-verification=false

# jwt
jwt.secret = ${jwtSecret}
jwt.access.token.expire = 3600000
jwt.refresh.token.expire = 144000000

#registration-kakao
spring.security.oauth2.client.registration.kakao.client-name=kakao
spring.security.oauth2.client.registration.kakao.client-id= ${id}
spring.security.oauth2.client.registration.kakao.client-secret= ${k
spring.security.oauth2.client.registration.kakao.redirect-uri=https
spring.security.oauth2.client.registration.kakao.authorization-gran
spring.security.oauth2.client.registration.kakao.scope=profile_nick
spring.security.oauth2.client.registration.kakao.client-authenticat

#provider-kakao
spring.security.oauth2.client.provider.kakao.authorization-uri=http
spring.security.oauth2.client.provider.kakao.user-name-attribute=id
spring.security.oauth2.client.provider.kakao.token-uri=https://kaut
spring.security.oauth2.client.provider.kakao.user-info-uri=https://

# mqtt
mqtt.broker.url=tcp://k11b107.p.ssafy.io:1883
mqtt.client.id=spring-mqtt-client

#smartthings
smartthings.secret = ${smartthings secret}

#redis set-key
```

```
spring.redis.set.key = priority
spring.redis.action.key = booleanAction
```

- front

```
VITE_API_URL=https://k11b107.p.ssafy.io/api
```

하드웨어

라즈베리 파이

> 라즈베리 파이는 창문 하나당 내부 감지용 센서 하나와 외부 감지용 센서 하나로 이루어져 있음.

- 라즈베리 파이 모델 : `라즈베리파이 4 Model B`
- 라즈베리 파이 OS : `Raspberry Pi OS (64-bit) released by 2024-10-22`
- 사용 센서
  - `nova pm sensor sds011`
  - `dht22`
  - `ccs811`
- 센서용 패키지

```
pip install paho-mqtt pyserial adafruit-circuitpython-dht adafruit-
```

- 센서 탐지용 코드

▼ main.py

```
import board
import time
import paho.mqtt.client as mqtt

from ccs811 import initialize_ccs811, read_ccs811
from pmsensor import initialize_serial, read_nova_pm
from dht22 import read_dht

broker_address = "k11b107.p.ssafy.io"

memberId = 1
```

```
windowsId = 4
placeId = 7
isInside = 0



def initialize_mqtt():
    client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1, f"ras
    client.connect(broker_address)
    return client



def main():
    client = initialize_mqtt()
    ser = initialize_serial()
    ccs811 = initialize_ccs811()
    last_humid = 20.0
    last_temp = 27.0
    last_co2 = 400
    last_tvoc = 0
    last_pm25 = 5.0
    last_pm10 = 5.0

    while True:
        pm25, pm10 = read_nova_pm(ser)
        co2, tvoc = read_ccs811(ccs811)
        print("co2 : ",co2," tvoc : ",tvoc)
        temp, humid = read_dht()

        if temp==None or humid == None :
            temp = last_temp

            humid = last_humid
        else :
            last_temp = temp
            last_humid = humid


        if pm25==None or pm10 == None:
            pm25 = last_pm25
            pm10 = last_pm10
        else :
            last_pm25 = pm25
```

```
            last_pm10 = pm10

        if co2==None or tvoc == None :
            co2 = last_co2
            tvoc = last_tvoc

        if co2>30000 or tvoc>30000 :
            co2 = last_co2
            tvoc = last_tvoc
            print('co2 chk')
        else :
            last_co2 = co2
            last_tvoc = tvoc

        print("CO2: {} PPM, TVOC: {} PPB".format(co2, tvoc))
        print(f"Temp: {temp:.1f}C, Humidity: {humid:.1f}%")
        print(f"PM2.5: {pm25} microgram , PM10: {pm10} microgram
        client.publish(f"sensor/{windowsId}", f"windowsId : {win
        pm25 : {pm25} pm10 : {pm10} co2 : {co2} tvoc : {tvoc} \
        temp : {temp} humid : {humid} placeId : {placeId} isInsi
        time.sleep(1)


if __name__ == "__main__":
    main()
```

▼ pmsensor.py

```
# nova_pm_sensor.py
import serial
import time

# Configure serial port (update with correct port name)
SERIAL_PORT = '/dev/ttyUSB0'  # Adjust this if needed after iden
BAUD_RATE = 9600

def initialize_serial():
    ser = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=2)
    return ser
```

```
def read_nova_pm(ser):
    if ser.isOpen():
        ser.flushInput()  # Clear input buffer
        data = ser.read(10)  # Read 10 bytes of data from Nova P
        if len(data) == 10 and data[0] == 0xAA and data[1] == 0x
            pm25 = data[2] + (data[3] << 8)  # PM2.5 concentrati
            pm10 = data[4] + (data[5] << 8)  # PM10 concentratio
            return pm25 / 10.0, pm10 / 10.0
        else:
            print("Failed to retrieve data from Nova PM sensor")
            return None, None
    else:
        print("Serial port not open")
        return None, None

#ser = initialize_serial()
#while True:
#   read_nova_pm(ser)
#   time.sleep(1)

# Example usage in main.py:
# from nova_pm_sensor import initialize_serial, read_nova_pm
# ser = initialize_serial()
# while True:
#     read_nova_pm(ser)
```

▼ dht22.py

```
# dht_sensor.py
import adafruit_dht
import board
import time

# DHT sensor setup
dht_device = adafruit_dht.DHT22(board.D14)

def read_dht():
    try:
        temperature = dht_device.temperature
        humidity = dht_device.humidity
        if humidity is not None and temperature is not None:
            return temperature, humidity
        else:
            print("Failed to retrieve data from humidity sensor"
```

```
            return None, None
    except RuntimeError as error:
        print(f"RuntimeError: {error.args[0]}. Retrying...")
        time.sleep(1)
        return None, None
#while True:
#   read_dht()
#   time.sleep(1)

# Example usage in main.py:
# from dht_sensor import read_dht
# while True:
#     temperature, humidity = read_dht()
#     time.sleep(2)
```

▼ ccs811.py

```
# ccs811_sensor.py
import time
import board
import adafruit_ccs811
import random

def initialize_ccs811():
    i2c = board.I2C()  # uses board.SCL and board.SDA
    ccs811 = adafruit_ccs811.CCS811(i2c)

    # Wait for the sensor to be ready
    while not ccs811.data_ready:
        pass

    return ccs811

def read_ccs811(ccs811):
    try:
        co2 = ccs811.eco2
        tvoc = ccs811.tvoc

    except:
        co2 = random.randint(400,500)
        tvoc = random.randint(0,50)

    return co2, tvoc
```

```
#ccs811 = initialize_ccs811()

#while True:
#   read_ccs811(ccs811)
#   time.sleep(1)


# Example usage in main.py:
# from ccs811_sensor import initialize_ccs811, read_ccs811
# ccs811 = initialize_ccs811()
# while True:
#     read_ccs811(ccs811)
#     time.sleep(0.5)
```

AI 코드 실행

- 실행해야하는 커맨드

```
wget https://github.com/conda-forge/miniforge/releases/latest/downl
bash Miniforge3-Linux-aarch64.sh

source ~/.bashrc

conda create -n window python=3.10 -y
conda activate window
pip install torch torchvision torchaudio

pip install tensorflow==2.10.0
pip install pandas numpy==1.23.5 scikit-learn shap tqdm gym
pip install PyYAML elasticsearch
```

- `AI/autoencoder/configs`에 config.yaml 파일 넣어주기
  - config.yaml

```
elasticsearch:
  host: "http://k11b107.p.ssafy.io:9200"
  username: "elastic"
  password: ${elasticsearchpassword}
  index_name: ${indexname}

springboot:
  windows_id: 4
```

```
home_id: 7
url: "https://k11b107.p.ssafy.io/api/actions"
open_url: "https://k11b107.p.ssafy.io/api/windows/open/auto/"
close_url: "https://k11b107.p.ssafy.io/api/windows/close/auto/"
status_url: "https://k11b107.p.ssafy.io/api/windows/state/"
```

- 그 후 build.sh를 실행시켜주면 AI 자동 실행

실행 방법

▼ 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-plugin`
- 이미 되있는건가 싶기도하고

▼ 젠킨스 설치

젠킨스 이미지 다운로드

- `sudo docker pull jenkins/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:latest`
  - **-d** : 컨테이너를 **데몬**으로 띄웁니다.(백그라운드)
  - **-p 8080:8080** : 컨테이너 외부와 내부 포트를 **포워딩**합니다. 좌측이 호스트 포트, 우측이 컨테이너 포트입니다.
  - **-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 jenkins만 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/$(. /etc/os-release
add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/
   $(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 없을 때
docker: 생성할 그룹의 이름입니다. 이 명령어는 docker라는 이름의 그룹을 만
usermod -aG docker jenkins

usermod: 사용자를 수정하는 명령어입니다.
-aG: 사용자를 새로운 그룹에 추가하는 옵션입니다. -a는 추가를 의미하고, -G
docker: 사용자가 추가될 그룹의 이름입니다.
jenkins: 그룹에 추가할 사용자의 이름입니다. 이 명령어는 jenkins 사용자를
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. 이 명령어
```

- docker-compose를 실행하기 위해 docker-compose 설치

```
curl -L "https://github.com/docker/compose/releases/download/v2.
그후 docker-compose -v 로 버전 확인
chmod +x docker-compose



curl -L "https://github.com/docker/compose/releases/download/v2.
chmod +x /usr/local/bin/docker-compose # 안해주면 실행x, which도 안뜸
ln -s /usr/local/bin/docker-compose /usr/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-recomme
    vim \
    apt-utils
RUN apt-get install ca-certificates curl gnupg lsb-release -y
RUN mkdir -p /etc/apt/keyrings
RUN curl -fsSL https://download.docker.com/linux/debian/gpg | gp
RUN echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/
RUN apt-get -y update
RUN apt-get install docker-ce docker-ce-cli containerd.io docker
RUN if [ -e /var/run/docker.sock ]; then chown jenkins:jenkins /

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-center/rootC

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

sudo docker restart jenkins
```

- 그후 `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

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] gitlab 연동 및 webhook 설정하기
jenkins gitlab 연동 및 webhook 설정

https://velog.io/@suhongkim98/jenkins-gitlab-연동-및-webhook-설정하기

velog

▼ 도커에 mysql 설치

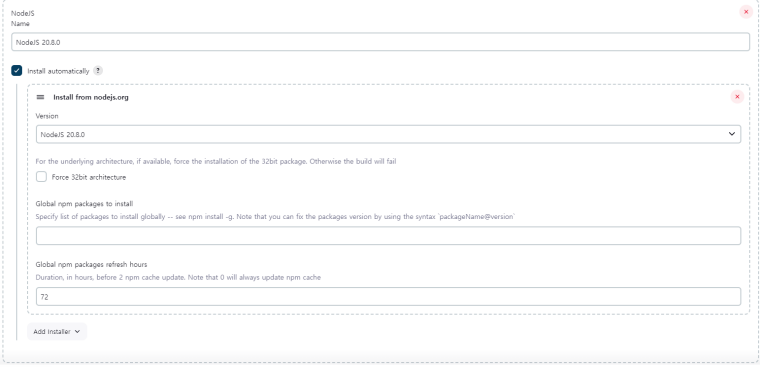
- `docker pull mysql`
- `sudo docker run --name mysql -e MYSQL_ROOT_PASSWORD='window11st1!2@aiot' -e MYSQL_DATABASE=windoorman -e TZ=Asia/Seoul -v /mysql/data:/var/lib/mysql -v /mysql/config:/etc/mysql/conf.d -v /mysql/init:/docker-entry-point-initdb.d -d -p 3306:3306 --network=windoorman mysql:latest`
  - -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 플러그인에서 nodeJS를 설치하자

Global Tool Configuration에서 Nodejs, Gradle 추가



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

> 💡 여기서 중요한 점은 깃랩이랑 연결하는건 api토큰이 맞지만 파이프라인에서 푸시, 풀 땡길 때는 퍼스널 액세스 토큰을 사용해야한다!

Git 스크립트 구성하기

```
pipeline {
    agent any

    tools {
        nodejs "nodejs"
        gradle "gradle"
    }

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

```
    }
}
```

- 위 명령어를 실행하면 해당 url을 clone받아온다.
- 클론받은 파일들은 jenkins_home에 workspace공간에 있음 (당연히 volume으로 연결한 호스트 디렉토리에도 존재한다)
- gitlab에서 clone할때 crendentail을 아이디 비밀번호로 만들고 아이디는 아이디, 비밀번호는 퍼스널액세스토큰을 받아서 넣어야한다.

▼ dev-BE 파이프라인

```
pipeline {
    agent any

    stages {
        stage('clone') {
            steps {
                git branch: 'dev-BE', credentialsId: '  gitlab_B3
            }
        }
        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: 'applica
                        script {
                            sh 'cp $applicationProperties src/mai
                            sh 'ls src/main/resources'
                            sh 'cat src/main/resources/applicatio
                        }
                    }
                }
            }
        }
        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 --networ
            }
        }
    }
}
```

▼ dev-FE 파이프라인

```
pipeline {
    agent any

    stages {
        stage('clone') {
            steps {
                git branch: 'dev-FE', credentialsId: '  gitlab_B3
            }
        }
        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/letsencrypt:/
            }
        }
```

```
    }
}
```

▼ develop 파이프라인

```
pipeline {
    agent any

    stages {
        stage('clone') {
            steps {
                git branch: 'develop', credentialsId: ' gitlab_B3
            }
        }
        stage('docker-compose down') {
            steps {
                sh 'docker-compose -f /var/jenkins_home/docker-co
            }
        }
        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: 'applica
                        script {
                            sh 'cp $applicationProperties src/mai
                            sh 'ls src/main/resources'
                            sh 'cat src/main/resources/applicatio
                        }
                    }
                }
            }
        }
        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/docker-co
            }
        }
    }
}
```

▼ dev-AI 파이프라인

```
pipeline {
    agent any

    stages {
        stage('clone') {
            steps {
                git branch: 'dev-AI', credentialsId: '  gitlab_B3
            }
        }
        stage('container down') {
            steps {
                sh 'docker stop supia-ai || true'
                sh 'docker rm supia-ai || true'
            }
        }
        stage('.env download') {
            steps {
                withCredentials([file(credentialsId: 'ai-env', va
                    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=8000 -p 8
            }
        }
    }
}
```

▼ 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 3000

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 /home/ubuntu/templates/logs:/var/log/nginx -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 파일이 위치
                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_add_x_fo
                proxy_set_header X-Forwarded-Proto $scheme;
                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 80;
    server_name k11b107.p.ssafy.io;
    return 308 https://k11b107.p.ssafy.io$request_uri;
}

server {
    listen 443 ssl;
    server_name k11b107.p.ssafy.io;
    ssl_certificate /etc/letsencrypt/live/p.ssafy.io/fullchain.p
    ssl_certificate_key /etc/letsencrypt/live/p.ssafy.io/privkey
#   location / {
#       proxy_pass http://front:5173;
#       add_header 'Cross-Origin-Embedder-Policy' 'require-corp'
#       add_header 'Cross-Origin-Opener-Policy' 'same-origin';
#   }
#   location /api {
#       charset utf-8;
#       proxy_pass http://back: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_
#       proxy_set_header X-Forwarded-Proto $scheme;
#   }
}
```

- 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 k11b107.p.ssafy.io;
        location / {

                root /usr/share/nginx/html/;
                index index.html index.htm;
                                try_files $uri /index.html;
```

```
                add_header 'Cross-Origin-Embedder-Policy' 'require-
                add_header 'Cross-Origin-Opener-Policy' 'same-origi
        }
}
```

▼ https SSL 설정

certbot 설치

1. `sudo apt-get install letsencrypt`
2. `sudo apt-get install certbot`
3. `sudo apt-get install certbot python3-certbot-nginx`

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

▼ 백엔드 도커파일 설정하기

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

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

▼ 프론트용 도커파일 설정

```
FROM node:20 as build-stage
WORKDIR /FE

RUN npm install -g pnpm

COPY package*.json pnpm-lock.yaml ./

RUN pnpm install

COPY . .
```

```
RUN pnpm run build

FROM nginx:stable-alpine as production-stage

#COPY conf/nginx.conf /etc/nginx/nginx.conf

COPY --from=build-stage /FE/dist /usr/share/nginx/html
COPY --from=build-stage /FE/dev-dist /usr/share/nginx/html

EXPOSE 5163

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

▼ 엘라스틱 서치 설정

엘라스틱 서치 설치

- `git clone` `https://github.com/deviantony/docker-elk.git`
- .env 파일 내의 환경 변수 값들을 수정해준다.
- `docker compose up setup`
- `docker compose up  -d`

elasticsearch.yml

```
## Default Elasticsearch configuration from Elasticsearch base imag
## https://github.com/elastic/elasticsearch/blob/main/distribution/
#
cluster.name: docker-cluster
network.host: 0.0.0.0
http.cors.enabled: true
http.cors.allow-origin: "*"
## X-Pack settings
## see https://www.elastic.co/guide/en/elasticsearch/reference/curr
#
xpack.license.self_generated.type: trial
xpack.security.enabled: true
```

logstash pipeline

```
input {
    beats {
        port => 5044
```

```
    }
    tcp {
        port => 50000
    }
}

filter {
    dissect {
        mapping => {
            "message" => "windowsId : %{windowsId} pm25 : %{pm25} pr
        }
    }
    mutate {
        convert => {
            "windowsId" => "integer"
            "pm25" => "float"
            "pm10" => "float"
            "co2" => "integer"
            "tvoc" => "integer"
            "temp" => "float"
            "humid" => "float"
            "placeId" => "integer"
            "isInside" => "integer"
        }
    }
    ruby {

        code => "event.set('kst_timestamp', event.get('@timestamp')
                 event.set('kst_index', event.get('@timestamp').time
                "


    }
}

output {
    stdout { codec => rubydebug }
    elasticsearch {
        hosts => "elasticsearch:9200"
        user => "elastic"
        password => ${elasticsearchpassword}
        index => "%{windowsId}-%{placeId}-%{kst_index}"
        }
}
```

filebeat config

```
logging.level: debug
logging.to_files: true
logging.files:
  path: /var/log/filebeat
  name: filebeat.log
  keepfiles: 7
  permissions: 0644


filebeat.config:
  modules:
    path: ${path.config}/modules.d/*.yml
    reload.enabled: false

filebeat.autodiscover:
  providers:
    - type: docker
      hints.enabled: true

filebeat.inputs:
- type: mqtt
  hosts:
    - tcp://k11b107.p.ssafy.io:1883
    - ssl://secure_broker:8883
  topics:
    - '#'

output.logstash:
  hosts: ["k11b107.p.ssafy.io:5044"]



processors:
- add_cloud_metadata: ~
```

▼ 레디스 설정

- `docker run -d -p 6379:6379 --name redis redis`

▼ mqtt 설정

- `docker run --name mosquitto -d -p 1883:1883 -v "./mosquitto/config:/mosquitto/config" eclipse-mosquitto`

💡 만약 mosquitto의 메세지를 저장하고 로그를 저장하고 싶으면 /mosquitto/data 와 /mosquitto/log 도 똑같이 volume 지정해주면 된다.

그리고 conf파일에

```
persistence true
persistence_location /mosquitto/data/
log_dest file /mosquitto/log/mosquitto.log
```

이렇게 설정해주면 됨

▼ **설명**

- `persistence true`
  - 이 옵션은 **브로커의 모든 메시지 상태를 디스크에 저장할지 여부**를 설정합니다. `true`로 설정하면 Mosquitto는 메시지 큐를 디스크에 저장하고, 브로커를 재시작해도 마지막 메시지 상태를 유지할 수 있습니다. 재시작 후에도 구독자에게 메시지를 지속적으로 전달할 수 있어서 유용합니다.
- `persistence_location /mosquitto/data/`
  - 메시지를 **저장할 파일 위치를 지정**하는 옵션입니다. `/mosquitto/data/` 폴더에 메시지 상태와 관련된 데이터를 저장하겠다는 의미입니다. 이 경로는 사용자가 설정한 경로에 따라 달라질 수 있으며, 지속적인 메시지 큐를 유지할 때 필요한 데이터가 이 위치에 저장됩니다.
- `log_dest file /mosquitto/log/mosquitto.log`
  - 이 옵션은 **로그를 파일에 저장할 위치**를 지정합니다. 여기서는 `/mosquitto/log/mosquitto.log` 파일에 로그 기록을 남기도록 설정한 것입니다. 이 로그 파일에는 MQTT 클라이언트 연결, 메시지 전송, 오류와 같은 Mosquitto의 운영 상태가 기록됩니다.

- 위 도커 명령어를 실행하기 전에 우분투에서 해당 config파일을 만들어줘야 한다.
  - `./mosquitto/config/mosquitto.conf`

```
allow_anonymous true
connection_messages true
log_type all
listener 1883
```

▼ 파이썬에서 메세지 올리기

패키지 설치

- `pip install paho.mqtt`

```
import paho.mqtt.client as mqtt

broker_address = "k11b107.p.ssafy.io"
client = mqtt.Client(mqtt.CallbackAPIVersion.VERSION1, "Raspberry
client.connect(broker_address) #연결

#메세지 발행하기
client.publish(f"sensor/supersonic/{memberId}", f"dist1 : {dist1}
```

▼ logstash 에서 메세지 받기

filebeat 설치

> 먼저 logstash에서 mqtt 형식의 메세지를 받으려면 filebeat를 거쳐야 한다

- `docker pull docker.elastic.co/beats/filebeat:8.15.3`
- configuration 파일 다운로드
  - `curl -L -O` `https://raw.githubusercontent.com/elastic/beats/8.15/deploy/docker/filebeat.docker.yml`
- config 설정 후 아래 명령어로 실행해주기
  - ```
    docker run -d \
    --name=filebeat \
    --user=root \
    --volume="$(pwd)/filebeat.docker.yml:/usr/share/filebeat/filebeat.yml:ro" \
    --volume="/var/lib/docker/containers:/var/lib/docker/containers:ro" \
    --volume="/var/run/docker.sock:/var/run/docker.sock:ro" \
    --volume="registry:/usr/share/filebeat/data:rw" \

    docker.elastic.co/beats/filebeat:8.15.3 filebeat -e --strict.perms=false
    ```
- filebeat conf 파일 설정

```
filebeat.config:
  modules:
    path: ${path.config}/modules.d/*.yml
    reload.enabled: false

filebeat.autodiscover:
  providers:
    - type: docker
      hints.enabled: true

filebeat.inputs:
```
```

```
- type: mqtt
  hosts:
    - tcp://k11b107.p.ssafy.io:1883
    - ssl://secure_broker:8883
  topics:
    - sensor/supersonic/1

output.logstash:
  hosts: ["k11b107.p.ssafy.io:5044"]



processors:
- add_cloud_metadata: ~
```

- 기본 logstash pipeline

```
input {
    beats {
        port => 5044
    }

    tcp {
        port => 50000
    }
}

## Add your filters / logstash plugins configuration here

output {
    elasticsearch {
        hosts => "elasticsearch:9200"
        user => "logstash_internal"
        password => "${LOGSTASH_INTERNAL_PASSWORD}"
    }
}
```

```
input {
        beats {
                port => 5044
```

```
        }

        tcp {
                port => 50000
        }
}

## Add your filters / logstash plugins configuration here

output {
        elasticsearch {
                hosts => "elasticsearch:9200"
                user => "elastic"
                password => "window11aiotdoorman"
                index => "test-%{+YYYY.MM.dd}"
        }
}
```

- exec 접속할때 -u root 로 접속하면 편하다