포팅메뉴얼

0. 개발 환경

🖥️ Backend

- JVM : `OpenJDK 17`
- Spring Boot : `3.0.6`
- Gradle : `7.6`
- IDE : IntelliJ

🖥️ Frontend

- Vue : `2.6.14`
- Vuetify : `2.6.0`

💾 DB

- MongoDB : `4.4.20`

🔗 Server & DevOps

- Amazon EC2
- Nginx : `1.18.0`
- Docker : `23.0.4`
- Docker-Compose : `1.24.1`
- Jenkins : `2.375.3`

🎯 모니터링툴 & 테스팅툴

- SonarQube : `10.0.0.68432`
- Prometheus : `2.44.0`
- Grafana : `9.5.1`

👥 협업툴

- GitLab
- Jira
- Notion
- MatterMost

1. Docker & Docker-Compose 설치

1) Docker 설치

```
sudo apt-get update

# 필수 패키지 설치
sudo apt-get install \
     apt-transport-https \
     ca-certificates \
     curl \
     gnupg \
     lsb-release

# GPG Key 인증
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

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

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

# 도커 확인
sudo service docker status
```

2) Docker-Compose 설치

```
# 설치
sudo curl -L "https://github.com/docker/compose/releases/download/1.24.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/
docker-compose

# 도커 컴포즈 파일을 실행 가능하게 하도록 권한을 부여
sudo chmod +x /usr/local/bin/docker-compose

# 도커 컴포즈 버전 확인
docker-compose -v
```

2. Jenkins 설치 및 설정

1) Jenkins 컨테이너 실행

```
# 설치 및 실행
sudo docker run -u 0 -d -p 9090:8080 -p 50000:50000 -v /var/jenkins:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock
 --name jenkins jenkins/jenkins:lts-jdk11

# 설치 확인
sudo docker images
```

2) Jenkins 접속 및 기본 설정

1. Jenkins 접속: `http://k8a501.p.ssafy.io:9090`
2. 비밀번호 입력: (EC2 서버에서 `sudo docker logs jenkins`로 확인 가능)
3. `Install suggested plugins` 클릭
4. 계정 설정 (계정명, 암호, 이름, 이메일)
5. 플러그인 설치
   - Dashboard → Jenkins 관리 → 플러그인 관리 → Available plugins
     - Gitlab 관련 항목 설치
       - `Gitlab`, `Generic Webhook Trigger`, `Gitlab API`, `Gitlab Authentication` 설치
     - Docker 관련 항목 설치
       - `Docker`, `Docker Commons`, `Docker Pipeline`, `Docker API` 설치
     - Gradle 관련 항목 설치
       - `gradle` 검색 시 나오는 모든 항목 설치
     - SonarQube 관련 항목 설치
       - `SonarQube Scanner for Jenkins` 설치
6. Gradle 설치
   - Jenkins 관리 → Global Tool Configuration → Gradle 설정
     - name: `WimojiGradle`, version: Gradle `7.6`
7. Jenkins 컨테이너 내부에 Docker 설치

```
# 젠킨스 컨테이너 실행
sudo docker exec -it jenkins bash
```

```
# linux 버전 확인
cat /etc/issue
# --------------- OS ---------------------------------
```

```
# root@DESKTOP-R4P59B3:/home/opendocs# cat /etc/issue
# Ubuntu 20.04.4 LTS \n \l
# --------------- jenkins Container OS -------------------------------
# root@DESKTOP-R4P59B3:/home/opendocs# docker exec -it jenkins /bin/bash
# root@8fc963af71bb:/# cat /etc/issue
# Debian GNU/Linux 11 \n \l

# Docker 설치
## - Old Version Remove
apt-get remove docker docker-engine docker.io containerd runc
## - Setup Repo
apt-get update
apt-get install \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /etc/apt/keyrings/docker.gpg
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/debian \
  $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null
## - Install Docker Engine
apt-get update
apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

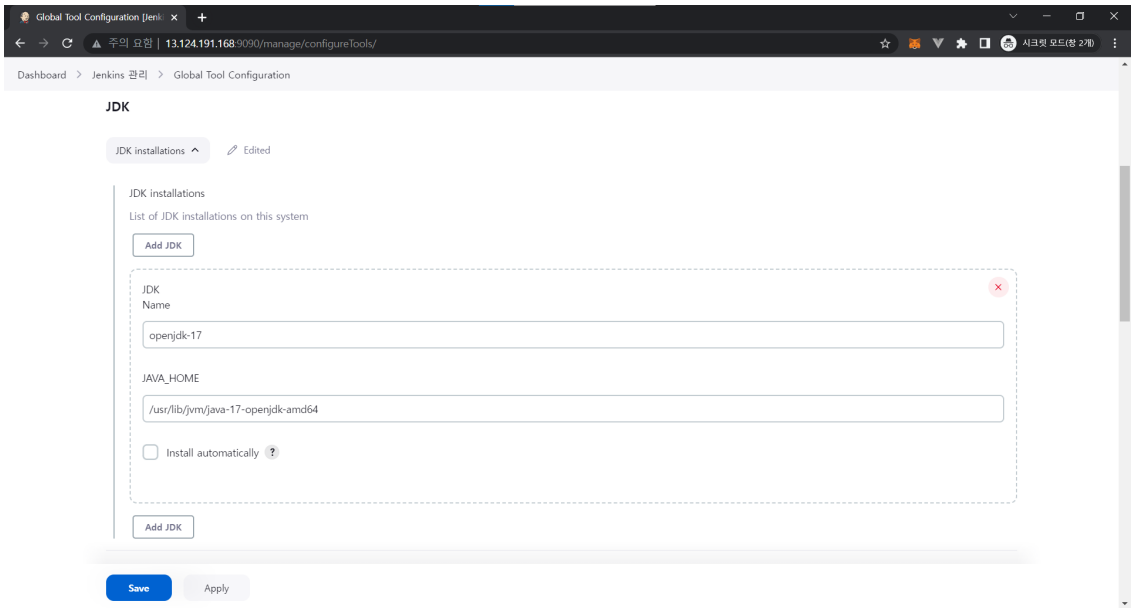
8. Jenkins 컨테이너 내부에 JDK 17 설치

```
# openjdk-17 설치
apt-get update
apt-get install openjdk-17-jdk

# JDK 폴더 위치를 검색하기 위해 mlocate 설치하고, 검색
apt-get install mlocate
updatedb
locate java | fgrep 17 | fgrep javac
# /usr/lib/jvm/java-17-openjdk-amd64

# JAVA_HOME 환경변수 변경 및 확인
export JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64
echo $JAVA_HOME
```

9. Jenkins 웹 사이트에서 JDK17 설정
   - Jenkins 관리 → Global Tool Configuration → JDK 설정
     - Name : `openjdk-17`, JAVA_HOME : `/usr/lib/jvm/java-17-openjdk-amd64`



3. MongoDB 설치

1) MongoDB 설치 및 실행

```
# MongoDB 공개 GPG 키를 가져오는 명령어를 입력
# 'OK' 응답이 나오면 정상적으로 키를 가져온 것
wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add -

# 만약, 'gnupg' 가 설치되지 않았다는 오류가 발생한다면, 다음 명령어를 입력하여 'gnupg' 설치
# 그리고 다시 키를 가져오는 명령어 입력
sudo apt install gnupg

# MongoDB를 설치하려는 Ubuntu 서버에 '/etc/apt/sources.list.d/mongodb-org-4.4.list' 파일 생성
# Ubuntu 버전 별로 다름 (현재는 Ubuntu 20.04)
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 multiverse" | sudo tee /etc/apt/sources.li
st.d/mongodb-org-4.4.list

# 로컬 패키지 데이터베이스를 업데이트하기
sudo apt update

# 최신 안정화 버전 설치
sudo apt-get install -y mongodb-org

# 만약, 'gnupg' 가 설치되지 않았다는 오류가 발생한다면, 다음 명령어를 입력하여 'gnupg' 설치
# 그리고 다시 키를 가져오는 명령어 입력
sudo apt install gnupg

# mongodb 실행
sudo service mongod start

# 아무런 에러 메세지가 뜨지 않는다면 잘 실행된 것
# mongodb 실행 확인 방법
sudo systemctl status mongod
```

2) 외부에서 접속 가능하도록 설정

- bindIP 를 0.0.0.0 으로 수정

```
# mongoDB는 디폴트로 내부에서만 접속을 허용하고 있으므로 설정해줘야 함
sudo vi /etc/mongod.conf
```

```
# for documentation of all options, see:
#   http://docs.mongodb.org/manual/reference/configuration-options/

# Where and how to store data.
storage:
  dbPath: /var/lib/mongodb
  journal:
    enabled: true
#  engine:
#  mmapv1:
#  wiredTiger:

# where to write logging data.
systemLog:
  destination: file
  logAppend: true
  path: /var/log/mongodb/mongod.log

# network interfaces
net:
  port: 27017
  bindIp: 0.0.0.0


# how the process runs
processManagement:
  timeZoneInfo: /usr/share/zoneinfo

#security:

#operationProfiling:
```

```
#replication:

#sharding:

## Enterprise-Only Options:

#auditLog:
```

```
# mongodb 재시작
sudo service mongod restart
```

4. Nginx 설정 및 SSL 인증서 발급

1) Nginx 설치

```
# 설치
sudo apt-get install nginx

# 설치 확인 및 버전 확인
nginx -v
```

2) letsencrypt 설치

```
# letsencrypt 설치
sudo apt update
sudo apt-get install letsencrypt

# 만약 nginx를 사용중이면 중지
sudo systemctl stop nginx
```

3) SSL 인증서 발급

```
# 인증서 발급
# sudo letsencrypt certonly --standalone -d [도메인]
sudo letsencrypt certonly --standalone -d k8a501.p.ssafy.io
# 자신의 이메일 쓰고 Agree
# 뉴스레터 no

# 인증서 위치 이동
sudo mv [인증서 위치 폴더] [이동할 인증서 위치 폴더]

# 인증서 위치 폴더로 이동
cd [인증서 위치 폴더]

# pem을 PKCS12 형식으로 변경
openssl pkcs12 -export -in fullchain.pem -inkey privkey.pem -out keystore.p12 -name airpageserver -CAfile chain.pem -caname root
```

4) Nginx 설정

- /etc/nginx/conf.d/default.conf와 /etc/nginx/sites-available/default 2개 파일 모두 동일하게 설정

```
sudo vi /etc/nginx/conf.d/default.conf
sudo vi /etc/nginx/sites-available/default
```

```
server {
        location /{
```

```
                proxy_connect_timeout      90;
                proxy_send_timeout         90;
                proxy_read_timeout         90;
                proxy_pass http://localhost:3000;
        }

        location /api {
                proxy_connect_timeout      90;
                proxy_send_timeout         90;
                proxy_read_timeout         90;
                proxy_set_header           Upgrade $http_upgrade;
                proxy_set_header           Connection "upgrade";
                proxy_pass http://localhost:8080/api;
        }

        listen 443 ssl;
        ssl_certificate [인증서 위치 폴더]/fullchain.pem;
        ssl_certificate_key [인증서 위치 폴더]/privkey.pem;
}

server {
        if ($host = k8a501.p.ssafy.io) {
                return 301 https://$host$request_uri;
        }

        listen 80;
        server_name k8a501.p.ssafy.io;
        return 404;
}
```

5) Nginx 실행

```
# nginx 실행
$ sudo systemctl start nginx

# 실행 확인
sudo systemctl status nginx
```

5. SonarQube 설치 및 설정

1) SonarQube 컨테이너 실행

```
sudo docker pull sonarqube
sudo docker run -d --name sonarqube -p 8000:9000 sonarqube
```

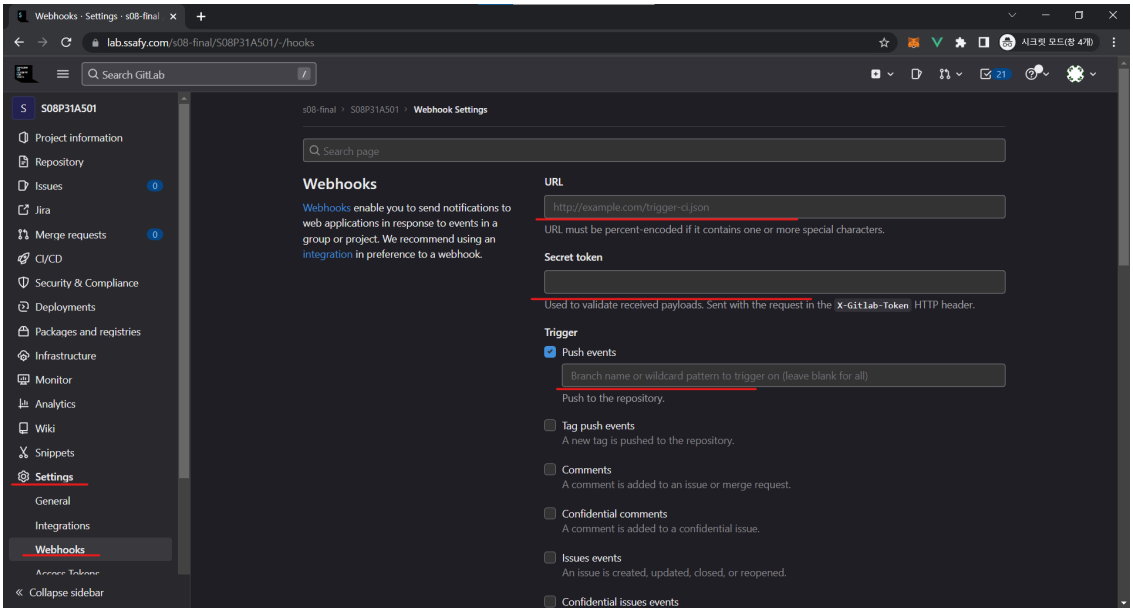
2) 프로젝트 생성

1. 웹 페이지 접속: `http://13.124.191.168:8000`

2. 프로젝트 생성
   - Create Projects → Project Display Name: `프로젝트명`, Main branch name: `sonarqube` → `Set Up`

3. Jenkins 설정
   a. `Jenkins 관리` → `Global Tool Configuration` → `Add SnarQube Scanner` → Name: `SonarQube Scanner` → `Install automatically` 체크 → Version: 최신 버전
   b. `Jenkins 관리` → `Configure System` → SonarQube servers → Name: `SonarQube servers` → URL: `http://13.124.191.168:8000` → `token` 설정 → `저장`

4. Jenkins와 Gitlab 연동
    a. `새로운 Item` → name: `sonarqube` → `Freestyle Project` → `OK`
    b. `소스 코드 관리` → `Git` → URL: `Gitlab 주소` → Credentials: `앞에서 설정한 계정` → branch: `sonarqube`
    c. `빌드 유발` → `Build when a change is pushed to Gitlab` → `고급` → Secret Token `Generate`
    d. Gitlab → `Settings` → `Webhooks` → 위에서 확인한 URL, Secret Token 입력 → branch: `sonarqube` → `Add webhook`



e. `Build Steps` → `Execute SonarQube Scanner` → 스크립트 작성

```
sonar.projectKey=[프로젝트 키]

# sonar.sources=springboot-framework, frontend/

sonar.sources=.
sonar.java.binaries=.


# sonar.sources=springboot-framework/user-service/src/main/java, springboot-framework/user-service/src/main/java/com/wimoj
i/controller, springboot-framework/emoji-service/src/main/java, springboot-framework/emoji-service/src/main/java/com/wimoj
i/controller, springboot-framework/chat-service/src/main/java

sonar.exclusions=**/TestController.java, **/TestService.java
```

6. Prometheus & Grafana 설치

1) Docker-Compose 파일 생성

```
mkdir docker-compose/grafana-prometheus
cd docker-compose/grafana-prometheus
vi docker-compose.yml
```

```
version: '3'
networks:
  monitor:
    driver: bridge

services:
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
```

```
    user: root
    volumes:
      - /home/monitor/prometheus/:/etc/prometheus/
      - /home/monitor/prometheus/data:/prometheus
    ports:
      - 8001:9090
    networks:
      - monitor
    restart: always
    user: root
  grafana:
    container_name: grafana
    image: grafana/grafana:latest
    environment:
      - GF_SECURITY_ADMIN_USER=[사용할 유저명]
      - GF_SECURITY_ADMIN_PASSWORD=[사용할 비밀번호]
      - GF_USERS_ALLOW_SIGN_UP=false
    volumes:
      - /home/monitor/grafana:/var/lib/grafana
      - /home/monitor/grafana/provisioning:/etc/grafana/provisioning
    ports:
      - 8002:3000
    depends_on:
      - prometheus
    networks:
      - monitor
    restart: always
    user: root
```

2) prometheus.yml 파일 생성

```
vi /home/monitor/prometheus/prometheus.yml
```

```
global:
  scrape_interval: 15s #scrape_interval은 어느 정도의 빈도로 prometheus가 대상들의 지표를 수집할것인지 설정한다.
  evaluation_interval: 15s #evaluation_interval 옵션은 prometheus가 rule 을 얼마나 자주 평가할것인지를 제어한다.

# prometheus 서버를 불러오는 규칙을 지정할 수 있다. 현재는 아무런 rule도 설정하지 않은 상태이다.
rule_files:

# scrape_configs 에서는 prometheus가 어떤 리소스를 모니터할것인지를 제어한다.
# prometheus는 HTTP enpoint로 자기 자신에 대한 데이터를 배포하기 때문에 자신의 동작 상태를 모니터하고 수집할 수 있다.
# default 설정 파일에서는 1개의 prometheus라고 불리는 1개의 job만 있는데 prometheus server가 배포한 시계열 데이터를 수집한다.
# job은 정적으로 구성된 localhost의 9090포트 target 하나로 구성되어 있다. prometheus는 target의 /metrics 경로를 통해 지표를 수집한다.
# 그래서 데이터를 수집하는 URL은 http://localhost:9090/metrics 가 된다.
scrape_configs:
  # The job name is added as a label `job=<job_name>` to any timeseries scraped from this config.
  - job_name: "prometheus"

    # metrics_path defaults to '/metrics'
    # scheme defaults to 'http'.

    static_configs:
    - targets: ["k8a501.p.ssafy.io:8001"]

  - job_name: 'user-service'
    scrape_interval: 59s
    metrics_path: '/api/user-service/actuator/prometheus'
    static_configs:
    - targets: ['k8a501.p.ssafy.io:8080']

  - job_name: 'emoji-service'
    scrape_interval: 59s
    metrics_path: '/api/emoji-service/actuator/prometheus'
    static_configs:
    - targets: ['k8a501.p.ssafy.io:8080']

  - job_name: 'chat-service'
    scrape_interval: 59s
    metrics_path: '/api/chat-service/actuator/prometheus'
    static_configs:
    - targets: ['k8a501.p.ssafy.io:8080']

  - job_name: 'gateway-service'
    scrape_interval: 59s
    metrics_path: '/actuator/prometheus'
    static_configs:
    - targets: ['k8a501.p.ssafy.io:8080']
```

3) Docker-Compose 실행

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

7. Jenkins와 Gitlab의 Webhook 연결

1. `새로운 Item` → 프로젝트 이름 입력 (API Gateway, User Service, Emoji Service, Chat Service 각각 생성) → `Freestyle project`

2. 소스 코드 관리
   → Git을 선택하고 Gitlab 주소 입력
   → Credentials 아래의 Add 버튼을 클릭해서 깃랩 아이디와 비밀번호 저장
   → 자동 배포될 브랜치 설정하기 (api-gateway, user-service, chat-service, emoji-service, discovery-service 각각 배포)

3. 빌드 유발 → `Build when a change is pushed to GitLab` 체크
   → `고급` → Secret token `Generate`
   → Gitlab webhook URL, Secret token 기억해두기

4. GitLab Settings → Webhooks → URL과 Secret token 입력
   → Push events: [원하는 브랜치] 설정 (release/user, release/chat 등) → Add Webhook
   → Test를 이용하여 원하는 테스트 가능

5. (Gradle 이용하는 컨테이너만 설정) Build Steps → `Invoke Gradle Script` → 생성해둔 Gradle 설정 → Tasks: `clean build` → 고급
   → Build File: `[build.gradle의 파일 경로]`

6. Add build Step → Execute Shell 작성
   - discovery-service (Spring Cloud - Eureka)

```
# discovery-service 컨테이너 생성
docker build -t discovery-service:latest ./springboot-framework/spring-cloud

# 이미 실행 중인 discovery-service 컨테이너가 있다면 중단하기
if (docker ps | grep discovery-service) then docker stop discovery-service; fi

# 빌드 실패로 남아있는 이미지들 삭제
docker image prune -f

# discovery-service 컨테이너 실행
docker run -d --rm --name discovery-service -p 8761:8080 discovery-service

# 중단한 뒤 남아있는 이미지들 삭제
docker image prune -f
```

   - gateway service

```
# gateway-service 컨테이너 생성
docker build -t gateway-service:latest ./springboot-framework/gateway-service

# 이미 실행 중인 gateway-service 컨테이너가 있다면 중단하기
if (docker ps | grep gateway-service) then docker stop gateway-service; fi
```

```
# 빌드 실패로 남아있는 이미지들 삭제
docker image prune -f

# gateway-service 컨테이너 실행
docker run -d --rm --name gateway-service -p 8080:8080 gateway-service

# 중단한 뒤 남아있는 이미지들 삭제
docker image prune -f
```

- user-service

```
# user-service 컨테이너 생성
docker build -t user-service:latest ./springboot-framework/user-service

# 이미 실행 중인 user-service 컨테이너가 있다면 중단하기
if (docker ps | grep user-service) then docker stop user-service; fi

# 빌드 실패로 남아있는 이미지들 삭제
docker image prune -f

# user-service 컨테이너 실행
docker run -d --rm --name user-service -p 8081:8080 user-service

# 중단한 뒤 남아있는 이미지들 삭제
docker image prune -f
```

- emoji-service
  - 포트는 지정하지 않음 (유레카에서 자동 지정)

```
# emoji-service 컨테이너 생성
docker build -t emoji-service:latest ./springboot-framework/emoji-service

# 이미 실행 중인 emoji-service 컨테이너가 있다면 중단하기
if (docker ps | grep emoji-service) then docker stop emoji-service; fi

# 빌드 실패로 남아있는 이미지들 삭제
docker image prune -f

# emoji-service 컨테이너 실행
docker run -d --rm --name emoji-service -p 8082:8080 emoji-service

# 중단한 뒤 남아있는 이미지들 삭제
docker image prune -f
```

- chat-service
  - 포트는 지정하지 않음 (유레카에서 자동 지정)

```
# chat-service 컨테이너 생성
docker build -t chat-service:latest ./springboot-framework/chat-service

# 이미 실행 중인 chat-service 컨테이너가 있다면 중단하기
if (docker ps | grep chat-service) then docker stop chat-service; fi

# 빌드 실패로 남아있는 이미지들 삭제
docker image prune -f

# chat-service 컨테이너 실행
docker run -d --rm --name chat-service -p 8083:8080 chat-service

# 중단한 뒤 남아있는 이미지들 삭제
docker image prune -f
```

- frontend

```
# frontend 컨테이너 생성
docker build -t frontend:latest ./frontend

# 이미 실행 중인 frontend 컨테이너가 있다면 중단하기
if (docker ps | grep frontend) then docker stop frontend; fi

# 빌드 실패로 남아있는 이미지들 삭제
docker image prune -f

# frontend 컨테이너 실행 (https 때문에 ssl 인증서가 있는 곳으로 마운트)
docker run -d --rm --name frontend -p 3000:3000 -v [인증서 위치 폴더]:[인증서를 넣을 폴더] -v /etc/localtime:/etc/localtime:ro fronte

# 중단한 뒤 남아있는 이미지들 삭제
docker image prune -f
```

7. 저장 후 지금 빌드

8. application.yml 및 .env 파일 설정

1) Jenkins 컨테이너 접속

```
# 젠킨스 컨테이너의 ID 확인
sudo docker ps -a

# 젠킨스 컨테이너 접속
sudo docker exec -it [컨테이너ID] bash
```

2) application.yml (Backend) 및 .env (Frontend) 생성

- Discovery Service (Spring Cloud - Eureka)

```
cd /var/jenkins_home/workspace/Discovery Service/springboot-framework/spring-cloud/src/main
mkdir resources
cd resources
vi application.yml
```

```
server:
  port: 8761

spring:
  application:
    name: spring-cloud # eureka에 등록될 id

eureka:
  client:
    register-with-eureka: false # default가 true client로서 등록되지 않도록 함
    fetch-registry: false
    service-url:
      defaultZone: http://k8a501.p.ssafy.io:8761/eureka/
```

- Gateway Service

```
cd /var/jenkins_home/workspace/Gateway\ Service/springboot-framework/gateway-service/src/main/
mkdir resources
cd resources
vi application.yml
```

```
server:
  port: 8080
  servlet:
```

```
    context-path: /api

spring:
  application:
    name: gateway-service
  cloud:
    gateway:
      default-filters:
        - DedupeResponseHeader=Access-Control-Allow-Origin Access-Control-Allow-Credentials
        - name: GlobalFilter
          args:
            preLogger: true
            postLogger: true
      routes:
        - id: user-service
          uri: lb://USER-SERVICE
          predicates:
            - Path=/api/user-service/**
        - id: emoji-service
          uri: lb://EMOJI-SERVICE
          predicates:
            - Path=/api/emoji-service/**
        - id: chat-service
          uri: lb://CHAT-SERVICE
          predicates:
            - Path=/api/chat-service/**


      globalcors:
        cors-configurations:
          '[/**]':
            allowedOriginPatterns: '*'
            allowedHeaders:
              - x-request-with
              - authorization
              - content-type
              - credential
              - X-AUTH-TOKEN
              - X-CSRF-TOKEN
            allowedMethods:
              - GET
              - POST
              - PUT
              - DELETE
              - OPTIONS

eureka:
  instance:
    hostname: k8a501.p.ssafy.io
    instanceId: ${spring.application.name}:${spring.application.instance_id:${random.value}}
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://k8a501.p.ssafy.io:8761/eureka/

management:
  endpoints:
    web:
      exposure:
        include: refresh, health, beans, busrefresh, info, prometheus, metrics, gateway
```

- User Service

```
cd /var/jenkins_home/workspace/User\ Service/springboot-framework/user-service/src/main/
mkdir resources
cd resources
vi application.yml
```

```
server:
  servlet:
    context-path: /api/user-service

spring:
  data:
    mongodb:
      host: k8a501.p.ssafy.io
      port: 27017
      database: testdb
```

```
  application:
    name: user-service

eureka:
  instance:
    prefer-ip-address: true
    hostname: k8a501.p.ssafy.io
    instanceId: ${spring.application.name}:${spring.application.instance_id:${random.value}}
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://k8a501.p.ssafy.io:8761/eureka/

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

jwt:
  expiration: [expiration]
  expirationRefresh: [expirationRefresh]
  secret1: [secret1]
  secret2: [secret2]
```

- Emoji Service

```
cd /var/jenkins_home/workspace/Emoji\ Service/springboot-framework/emoji-service/src/main/
mkdir resources
cd resources
vi application.yml
```

```
server:
  servlet:
    context-path: /api/emoji-service

spring:
  data:
    mongodb:
      host: k8a501.p.ssafy.io
      port: 27017
      database: testdb

  application:
    name: emoji-service

eureka:
  instance:
    prefer-ip-address: true
    hostname: k8a501.p.ssafy.io
    instanceId: ${spring.application.name}:${spring.application.instance_id:${random.value}}
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://k8a501.p.ssafy.io:8761/eureka

management:
  endpoints:
    web:
      exposure:
        include: refresh, health, beans, busrefresh, info, prometheus, metrics
```

- Chat Service

```
cd /var/jenkins_home/workspace/Chat\ Service/springboot-framework/chat-service/src/main/
mkdir resources
cd resources
vi application.yml
```

```
server:
  servlet:
    context-path: /api/chat-service

spring:
  data:
    mongodb:
      host: k8a501.p.ssafy.io
      port: 27017
      database: testdb

  application:
    name: chat-service

eureka:
  instance:
    prefer-ip-address: true
    hostname: k8a501.p.ssafy.io
    instanceId: ${spring.application.name}:${spring.application.instance_id:${random.value}}
  client:
    register-with-eureka: true
    fetch-registry: true
    service-url:
      defaultZone: http://k8a501.p.ssafy.io:8761/eureka

management:
  endpoints:
    web:
      exposure:
        include: refresh, health, beans, busrefresh, info, prometheus, metrics
```

- .env (Frontend)

```
cd /var/jenkins_home/workspace/Frontend/frontend
vi .env
```

```
# 백엔드 서버 URL
VUE_APP_API_SERVICE_URL=[백엔드 서버 URL]
# 카카오 로컬 API 주소
VUE_APP_KAKAOMAP_BASE_URL=[카카오 로컬 API 주소]
# 카카오 API 키
VUE_APP_KAKAOMAP_API_KEY=[카카오 API 키]
```

3) Jenkins에서 각각의 컨테이너 다시 빌드하기