



포팅 메뉴얼

☰ 태그

📁 자료함

1. 개발환경

[1.1 Frontend](#)

[1.2 Backend](#)

[1.3 Server](#)

[1.4 Database](#)

[1.5 UI/UX](#)

[1.6 Mobile](#)

[1.7 IDE](#)

[1.8 형상 / 이슈 관리](#)

2. 인프라 세팅

2.1 서버 세팅 공통

[2.1.0 서버 시간대 통일 / 깃 정보 등록](#)

[2.1.1 Zulu 17 설치](#)

[2.1.2 NGINX 설치](#)

[2.1.3 SSL 인증서 등록 \(Jenkins 제외\)](#)

2.2 Jenkins 서버 세팅

[2.2.1 Jenkins 설치](#)

[2.2.2 NGINX 설정](#)

[2.2.3 Maven 설치](#)

[2.2.4 Docker 설치](#)

[2.2.5 젠킨스 서버 방화벽 설정](#)

2.3 백엔드 배포 서버 세팅

[2.3.1 Maven 설치](#)

[2.3.2 Redis 설치 & 추가](#)

[2.3.3 MySQL 설치 / 설정](#)

[2.3.4 MongoDB 설치 / 세팅](#)

[2.3.5 Docker 설치](#)

[2.3.6 BE 배포 서버 NGINX 세팅](#)

[2.3.7 방화벽 설정](#)

[2.3.8 RabbitMQ 설치](#)

[2.3.9 Zookeeper & Kafka 설치](#)

[2.3.10 Kafka Connect 설치](#)

[2.3.11 JDBC Connector 설치](#)

- 2.3.12 Kafka Connect 서비스 등록
- 2.3.13 MySQL Source Connector 등록
- 2.3.14 MySQL Sink Connector 등록
- 2.3.15 Connector 등록 확인
- 2.4 프론트 배포 서버 세팅
 - 2.4.1 프론트 서버 방화벽 설정
 - 2.4.2 npm 설치
 - 2.4.3 프론트 NGINX 세팅
- 3. Jenkins Pipeline
 - 3.1 BE 배포 세팅
 - 3.1.1 BE 빌드 후 배포
 - 3.1.2 Spring Cloud Config 갱신
 - 3.2 FE 배포 세팅
- 4. Front 세팅
 - *npm 패키지 이름 규칙
 - 1 vite + react + ts 설치
 - 2 pwa 설치
 - 3 tailwindcss + styled-components + twin.macro 설치
 - 파일 수정
 - 4 zustand 설치
 - 5 TanStack Query 설치
 - 6 prettier, eslint 설치
 - 1. Prettier와 ESLint 관련 플러그인을 설치
 - 2. **eslint.config.js** 수정
 - 2. **.prettierrignore** 파일 생성
 - 3. **.prettierrc** 파일 생성
 - 4. VS code extension 에서 아래 설치
 - 5. VS Code 에서 ctrl + shift + p → settings.json 에 추가
 - 6. ctrl + s 눌러서 저장시키면 변환되는 것을 볼 수 있음
 - 7 ngrok 설치 - 로컬에서 http → https 변환용 (배포 전)
 - 다운받은 파일의 압축을 해제하고 ngrok.exe를 관리자 권한으로 실행
 - 1. Ngrok 계정 생성
 - 2. 인증 토큰 확인
 - 3. 로컬 환경에 Ngrok 인증 등록
 - 4. Ngrok 다시 실행
 - *기존 인증 토큰 재설정
 - 5. **vite.config.js** 수정
 - Forwarding 주소로 접근
 - *크롬에서 주소창에 앱 설치 아이콘이 안 보일 때
 - 8 실행
 - 9 절대 경로 설정

- [1. vite.config.ts](#)
 - [2. tsconfig.json](#)
 - [3. tsconfig.app.json](#)
- [10 라우터 설치](#)
- [1 1 Shadcn 설치 \(UI 라이브러리\)](#)
- [1 2 axios 설치](#)
- [1 3 react cookie 설치](#)
- [4. 엘라스틱 서치 세팅](#)
 - [jaso-analyzer, nori tokenizer 설치](#)
 - [logstash 세팅](#)
- [5. 백엔드 세팅](#)
 - [5.1 Config 서버](#)
 - [5.2 Eureka 서버](#)
 - [5.3 Gateway 서버](#)
 - [5.4 Auth 서버](#)
 - [5.5 Product 서버](#)
 - [5.6 Feed 서버](#)
 - [5.7 Mypage 서버](#)
 - [5.8 Financial 서버](#)
 - [5.9 Board 서버](#)
 - [5.10 Chat 서버](#)
 - [5.11 Search 서버](#)
- [6. 외부 서비스](#)
 - [6.1 소셜 로그인 - Kakao](#)
 - [6.1.1 카카오 로그인 활성화](#)
 - [6.1.2 리다이렉트 UR설정](#)
 - [6.1.3 요청 흐름](#)

1. 개발환경

1.1 Frontend

- React 18.3.1
- zustand 5.0.3
- TailWind CSS 3.4.17
- TypeScript 5.7.2
- Vite 6.2.0
- TanStack Query

- 라이브러리
 - twin.macro(Tailwind CSS v3 + @emotion)
 - react router dom
 - react-icons, iconoir-react
 - eslint
 - prettier
 - postcss

1.2 Backend

- Java
 - Azul Zulu 17.0.14
 - Spring Boot 3.4.2, 3.4.4
 - Spring Cloud
 - Config
 - Gateway
 - Eureka
 - Lombok
 - JWT
 - Oauth 2.0
 - STOMP
 - Maven 3.9.9

1.3 Server

Ubuntu 22.04

Docker

Let'sEncrypt (SSL)

1.4 Database

- MySQL 8.0.41

- MongoDB 6.0.20
- Redis 7.4.2
- AWS S3

1.5 UI/UX

- Figma, Canva

1.6 Mobile

- PWA
- Expo
 - React Native Webview

1.7 IDE

- VS Code
- IntelliJ IDEA

1.8 형상 / 이슈 관리

- GitLab
- Jira
- Github

2. 인프라 세팅

2.1 서버 세팅 공통

2.1.0 서버 시간대 통일 / 깃 정보 등록

```
# 서버 시간대 확인  
timedatectl
```

```
# 서울이 아닐 경우
sudo timedatectl set-timezone Asia/Seoul
```

```
git config --global user.name ""
git config --global user.email ""
```

2.1.1 Zulu 17 설치

```
sudo apt install gnupg ca-certificates curl
# 키 등록
curl -s https://repos.azul.com/azul-repo.key | sudo gpg --dearmor -o /usr/
share/keyrings/azul.gpg
echo "deb [signed-by=/usr/share/keyrings/azul.gpg] https://repos.azul.co
m/zulu/deb stable main" | sudo tee /etc/apt/sources.list.d/zulu.list

sudo apt-get update
sudo apt-get install -y zulu17-jdk

# 설치된 자바 폴더 위치 확인 (zulu 단독 설치 시)
sudo update-alternatives --config java
# 나온 경로 복사하여 JAVA_HOME 환경변수 설정해주기
# /usr/lib/jvm/zulu17/bin/java

sudo nano /etc/environment
PATH=":/usr/lib/jvm/zulu17/bin" # 기존에 추가
JAVA_HOME="/usr/lib/jvm/zulu17"
# Ctrl + O , Ctrl + X 로 저장 후 나가기

# 변경사항 적용
source /etc/environment
# 적용 확인
echo $JAVA_HOME
```

2.1.2 NGINX 설치

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

```
sudo systemctl enable nginx
sudo systemctl start nginx
```

2.1.3 SSL 인증서 등록 (Jenkins 제외)

```
sudo apt-get update
sudo apt-get install -y certbot python3-certbot-nginx
```

```
sudo systemctl stop nginx
```

```
# 적용할 도메인 주소와 이메일 입력해주기
sudo certbot certonly --nginx -d {도메인주소}
```

```
# 생성 후 nginx에 키 등록
sudo nano /etc/nginx/sites-available/default
```

```
server {
    listen 443 ssl;
    server_name {도메인 주소};

    client_max_body_size 50M;
    proxy_set_header Connection keep-alive;
    keepalive_timeout 65;

    ssl_certificate /etc/letsencrypt/live/{도메인주소}/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/{도메인주소}/privkey.pem;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers 'ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128
-GCM-SHA256:DHE-RSA-AES128-GCM-SHA256';
    ssl_prefer_server_ciphers on;

    이후 세팅...
...
}
```

2.2 Jenkins 서버 세팅

2.2.1 Jenkins 설치

```
# jenkins 설치
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins -y

# 권한 재확인
sudo chown -R jenkins:jenkins /var/lib/jenkins
sudo chown -R jenkins:jenkins /var/cache/jenkins
sudo chown -R jenkins:jenkins /var/log/jenkins

# 서비스 등록 및 실행
sudo systemctl enable jenkins
sudo systemctl start jenkins
```

2.2.2 NGINX 설정

```
sudo nano /etc/nginx/sites-available/jenkins
sudo ln -s /etc/nginx/sites-available/jenkins /etc/nginx/sites-enabled/
sudo rm /etc/nginx/sites-enabled/default
sudo systemctl restart nginx
```

앞단에 프록시 서버가 하나 있어서 거기에 맞게 NGINX 설정 - 80 포트로만 받게 설정됨

```
# /etc/nginx/sites-available/jenkins

upstream jenkins {
    server 127.0.0.1:8080;
    keepalive 32; # 커넥션 유지
}
```



```

server {
    listen 80;
    server_name localhost;

    ignore_invalid_headers off;

    location / {
        proxy_pass http://jenkins;

        # 프록시 헤더 설정
        proxy_set_header Host $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 https; # 앞단의 HTTPS를 위해
        proxy_set_header X-Forwarded-Host $host;
        proxy_set_header X-Forwarded-Port 443;    # HTTPS 포트

        # 리다이렉션 설정
        proxy_redirect http://jenkins https://$host;

        proxy_http_version 1.1;
        proxy_request_buffering off;
        proxy_buffering off;

        # 타임아웃 설정
        proxy_connect_timeout 150;
        proxy_send_timeout 100;
        proxy_read_timeout 100;
    }
}

```

2.2.3 Maven 설치

```

sudo mkdir /usr/local/maven
cd /usr/local/maven

sudo wget https://archive.apache.org/dist/maven/maven-3/3.9.9/binaries/a

```

```

apache-maven-3.9.9-bin.tar.gz
sudo tar -xvzf apache-maven-3.9.9-bin.tar.gz

sudo nano /etc/environment
PATH=":/usr/local/maven/apache-maven-3.9.9/bin" #기존에 추가
MAVEN_HOME="/usr/local/maven/apache-maven-3.9.9"
CLASSPATH=""

source /etc/environment

echo $MAVEN_HOME # 설치 확인

```

2.2.4 Docker 설치

```

sudo apt-get update
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/
keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/doc
ker.asc] https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-pl
ugin docker-compose-plugin

# 일반 사용자 권한 설정
sudo usermod -aG docker $USERa

# 등록 && 시작
sudo systemctl enable docker
sudo systemctl start docker

```

2.2.5 젠킨스 서버 방화벽 설정

```
sudo ufw allow OpenSSH
sudo ufw allow 80 # HTTP
sudo ufw allow 443 # HTTPS
sudo ufw enable
```

2.3 백엔드 배포 서버 세팅

2.3.1 Maven 설치

```
sudo mkdir /usr/local/maven
cd /usr/local/maven

sudo wget https://archive.apache.org/dist/maven/maven-3/3.9.9/binaries/apache-maven-3.9.9-bin.tar.gz
sudo tar -xvzf apache-maven-3.9.9-bin.tar.gz

sudo nano /etc/environment
PATH="/usr/local/maven/apache-maven-3.9.9/bin" #기존에 추가
MAVEN_HOME="/usr/local/maven/apache-maven-3.9.9"
CLASSPATH="."

source /etc/environment

echo $MAVEN_HOME # 설치 확인
```

2.3.2 Redis 설치 & 추가

```
sudo apt-get update
sudo apt-get install -y redis-server

# 서비스 등록 및 실행
sudo systemctl start redis
sudo systemctl enable redis
```

```
# 접속
redis-cli

# 키 조회
keys *
# get "조회할 키 이름"

# 서버 하나 더 열기
sudo cp /etc/redis/redis.conf /etc/redis/redis2.conf
sudo nano /etc/redis/redis2.conf
# 수정
port 6380
pidfile /run/redis/redis-server2.pid
logfile /var/log/redis/redis-server2.log
dbfilename dump2.rdb
dir /var/lib/redis2

cp /lib/systemd/system/redis-server.service /lib/systemd/system/redis-server2.service
sudo nano /lib/systemd/system/redis-server2.service
# 수정
[Unit]
Description=Advanced key-value store second
ExecStart=/usr/bin/redis-server /etc/redis/redis2.conf
ReadWriteDirectories=-/var/lib/redis2

[Install]
WantedBy=multi-user.target
Alias=redis2.service

sudo systemctl daemon-reload
sudo systemctl enable redis-server2.service
sudo systemctl start redis-server2.service
sudo systemctl status redis-server2.service

# 접속
redis-cli -p 6380
```

2.3.3 MySQL 설치 / 설정

```
# 설치
sudo apt-get update && sudo apt-get install -y mysql-server

# 포트 허용
sudo ufw allow mysql

# 시작 등록 및 시작
sudo systemctl enable mysql
sudo systemctl start mysql

# mysql 접속
sudo mysql

# 루트유저 비밀번호 생성
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY '비밀번호';

# 권한 갱신
FLUSH PRIVILEGES;

# 유저 생성
CREATE USER '유저명'@'localhost' IDENTIFIED BY '유저비번';

# 데이터베이스 생성 후
CREATE DATABASE 새로운데이터베이스;
SHOW DATABASES;

# 데이터베이스 권한 주기
GRANT ALL PRIVILEGES ON 생성한DB.* TO '권한 줄 유저명'@'localhost';
FLUSH PRIVILEGES; # 권한 갱신

EXIT;

# 이후엔 mysql -u 유저명 -p 로 sql 접속
```

2.3.4 MongoDB 설치 / 세팅

```
# MongoDB 공개 GPG키 가져오기
sudo apt-get install gnupg curl
```

```

curl -fsSL https://www.mongodb.org/static/pgp/server-6.0.asc | \
  sudo gpg -o /usr/share/keyrings/mongodb-server-6.0.gpg \
  --dearmor

# 사용중인 우분투 버전에 맞게 목록파일 생성 - 22.04(jammy)
echo "deb [ arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-
server-6.0.gpg ] https://repo.mongodb.org/apt/ubuntu jammy/mongodb-or
g/6.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-6.0.list

# 최신 릴리즈 설치
sudo apt-get update && sudo apt-get install -y mongodb-org

# 등록 && 실행
sudo systemctl enable mongod
sudo systemctl start mongod

# MongoDB 접속
mongo

# 데이터베이스 생성
use 데이터베이스

# 테스트 데이터 생성
db.test.insertOne({ "name": "test" })

# db 목록 확인
show dbs

# collections 확인
show collections

# collection 내부 데이터 확인
db.[collection 이름].find()

# 데이터 입력
db.[collection 이름].insert<Many-여러개, One-1개>(데이터)

```

```
# 데이터 삭제
```

```
db.[collection 이름].delete<Many-여러개, One-1개, All-전부>(데이터)
```

2.3.5 Docker 설치

```
sudo apt-get update
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/
keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/doc
ker.asc] https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

```
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-pl
ugin docker-compose-plugin
```

```
# 일반 사용자 권한 설정
```

```
sudo usermod -aG docker $USERa
```

```
# 등록 && 시작
```

```
sudo systemctl enable docker
sudo systemctl start docker
```

2.3.6 BE 배포 서버 NGINX 세팅

```
공통 세팅
```

```
...
```

```
location / {
    proxy_pass http://localhost:8080/; # Gateway 서비스 주소
    proxy_set_header Host $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;
}

location /chat/ {
    proxy_pass http://localhost:8082/chat/; # WebSocket 서비스 주소 (직접
연결)
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "Upgrade";
    proxy_set_header Host $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;

    client_max_body_size 50M;
    proxy_read_timeout 300;
    proxy_send_timeout 300;
}

location /config/ {
    proxy_pass http://localhost:8888/; # Config Server 포트
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
}

location /rabbitmq/ {
    proxy_pass http://localhost:15672/; # RabbitMQ 관리 UI 포트
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
}

location /eureka/ {
    proxy_pass http://localhost:8761/; # 유레카 포트
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;

    sub_filter 'href="/" 'href="/eureka/';

```



```
# uri 매핑 하면서 꼬인 WebUI 정적리소스 위치 잡아주기
sub_filter_once off;
}

location = /50x.html {
    root /usr/share/nginx/html;
}
}
```

2.3.7 방화벽 설정

```
sudo ufw allow OpenSSH
sudo ufw allow 80 # HTTP
sudo ufw allow 443 # HTTPS
sudo ufw allow 8080 # gateway
sudo ufw allow 8888 # config
sudo ufw allow 5601 # kibana
sudo ufw allow 8082 # chatting
sudo ufw enable
```

2.3.8 RabbitMQ 설치

```
# Rabbitmq 공식 설치 방법으로 설치
sudo apt-get install curl gnupg apt-transport-https -y

## Team RabbitMQ's main signing key
curl -1sLf "https://keys.openpgp.org/vks/v1/by-fingerprint/0A9AF2115F4687BD29803A206B73A36E6026DFCA" | sudo gpg --dearmor | sudo tee /usr/share/keyrings/com.rabbitmq.team.gpg > /dev/null
## Community mirror of Cloudsmith: modern Erlang repository
curl -1sLf https://github.com/rabbitmq/signing-keys/releases/download/3.0/cloudsmith.rabbitmq-erlang.E495BB49CC4BBE5B.key | sudo gpg --dearmor | sudo tee /usr/share/keyrings/rabbitmq.E495BB49CC4BBE5B.gpg > /dev/null
## Community mirror of Cloudsmith: RabbitMQ repository
curl -1sLf https://github.com/rabbitmq/signing-keys/releases/download/3.0/cloudsmith.rabbitmq-server.9F4587F226208342.key | sudo gpg --dear
```

```
mor | sudo tee /usr/share/keyrings/rabbitmq.9F4587F226208342.gpg > /dev/null
```

```
## Add apt repositories maintained by Team RabbitMQ
```

```
sudo tee /etc/apt/sources.list.d/rabbitmq.list <<EOF
```

```
## Provides modern Erlang/OTP releases
```

```
##
```

```
deb [arch=amd64 signed-by=/usr/share/keyrings/rabbitmq.E495BB49CC4BBE5B.gpg] https://ppa1.rabbitmq.com/rabbitmq/rabbitmq-erlang/deb/ubuntu jammy main
```

```
deb-src [signed-by=/usr/share/keyrings/rabbitmq.E495BB49CC4BBE5B.gpg] https://ppa1.rabbitmq.com/rabbitmq/rabbitmq-erlang/deb/ubuntu jammy main
```

```
# another mirror for redundancy
```

```
deb [arch=amd64 signed-by=/usr/share/keyrings/rabbitmq.E495BB49CC4BBE5B.gpg] https://ppa2.rabbitmq.com/rabbitmq/rabbitmq-erlang/deb/ubuntu jammy main
```

```
deb-src [signed-by=/usr/share/keyrings/rabbitmq.E495BB49CC4BBE5B.gpg] https://ppa2.rabbitmq.com/rabbitmq/rabbitmq-erlang/deb/ubuntu jammy main
```

```
## Provides RabbitMQ
```

```
##
```

```
deb [arch=amd64 signed-by=/usr/share/keyrings/rabbitmq.9F4587F226208342.gpg] https://ppa1.rabbitmq.com/rabbitmq/rabbitmq-server/deb/ubuntu jammy main
```

```
deb-src [signed-by=/usr/share/keyrings/rabbitmq.9F4587F226208342.gpg] https://ppa1.rabbitmq.com/rabbitmq/rabbitmq-server/deb/ubuntu jammy main
```

```
# another mirror for redundancy
```

```
deb [arch=amd64 signed-by=/usr/share/keyrings/rabbitmq.9F4587F226208342.gpg] https://ppa2.rabbitmq.com/rabbitmq/rabbitmq-server/deb/ubuntu jammy main
```

```
deb-src [signed-by=/usr/share/keyrings/rabbitmq.9F4587F226208342.gpg] https://ppa2.rabbitmq.com/rabbitmq/rabbitmq-server/deb/ubuntu jammy main
```

EOF

Update package indices

```
sudo apt-get update -y
```

Install Erlang packages

```
sudo apt-get install -y erlang-base \
                        erlang-asn1 erlang-crypto erlang-eldap erlang-ftp erlang-ine
ts \
                        erlang-mnesia erlang-os-mon erlang-parsetools erlang-publi
c-key \
                        erlang-runtime-tools erlang-snmp erlang-ssl \
                        erlang-syntax-tools erlang-tftp erlang-tools erlang-xmerl
```

Install rabbitmq-server and its dependencies

```
sudo apt-get install rabbitmq-server -y --fix-missing
```

```
sudo systemctl start rabbitmq-server
```

```
sudo systemctl enable rabbitmq-server
```

관리 플러그인 활성화

```
sudo rabbitmq-plugins enable rabbitmq_management
```

#ubuntu User can only log in via localhost

외부 접속하면 guest로 로그인 불가이므로 유저 만들어줘야 함 / 관리자권한 부여

```
sudo rabbitmqctl add_user 유저명 비밀번호
```

```
sudo rabbitmqctl set_user_tags 유저명 administrator
```

```
sudo rabbitmqctl set_permissions -p / 유저명 ".*" ".*" ".*"
```

2.3.9 Zookeeper & Kafka 설치

kafka 다운로드 (3.7.0 버전 사용)

```
cd /opt
```

```
sudo wget https://archive.apache.org/dist/kafka/3.7.0/kafka_2.13-3.7.0.tgz
```

압축 해제 후 폴더명 정리

```
sudo tar -xvzf kafka_2.13-3.7.0.tgz
```

```
sudo mv kafka_2.13-3.7.0 kafka
```

환경변수 등록

```
export PATH=$PATH:/opt/kafka/bin
```

서비스 생성 및 등록

```
sudo nano /etc/systemd/system/zookeeper.service
```

```
#/etc/systemd/system/zookeeper.service
```

```
[Unit]
```

```
Description=Zookeeper Service
```

```
After=network.target
```

```
[Service]
```

```
Type=simple
```

```
ExecStart=/opt/kafka/bin/zookeeper-server-start.sh /opt/kafka/config/zoo  
keeper.properties
```

```
ExecStop=/opt/kafka/bin/zookeeper-server-stop.sh
```

```
Restart=on-failure
```

```
[Install]
```

```
WantedBy=multi-user.target
```

```
sudo nano /etc/systemd/system/kafka.service
```

```
#/etc/systemd/system/kafka.service
```

```
[Unit]
```

```
Description=Apache Kafka Server
```

```
After=network.target zookeeper.service
```

```
Requires=zookeeper.service
```

```
[Service]
```

```
Type=simple
```

```
User=root
```

```
ExecStart=/opt/kafka/bin/kafka-server-start.sh /opt/kafka/config/server.pr  
operties
```

```
ExecStop=/opt/kafka/bin/kafka-server-stop.sh
```

```
Restart=on-failure
```

```
[Install]
```

```
WantedBy=multi-user.target
```

```
# 서비스 등록 및 실행
```

```
sudo systemctl daemon-reload
```

```
sudo systemctl enable zookeeper
```

```
sudo systemctl start zookeeper
```

```
sudo systemctl enable kafka
```

```
sudo systemctl start kafka
```

2.3.10 Kafka Connect 설치

```
# 작업 디렉토리로 이동
```

```
cd /opt/kafka
```

```
# Confluent Hub 클라이언트 다운로드
```

```
wget http://client.hub.confluent.io/confluent-hub-client-latest.tar.gz
```

```
# 압축 해제
```

```
tar -xvf confluent-hub-client-latest.tar.gz
```

```
# 환경변수 추가
```

```
sudo nano /etc/environment
```

```
PATH=":/opt/kafka/bin" # 맨 뒤에 추가
```

```
CONFLUENT_HOME="/opt/kafka"
```

```
source /etc/environment
```

```
# kafka connect 설정파일
```

```
sudo nano /opt/kafka/config/connect-distributed.properties
```

```
# /opt/kafka/config/connect-distributed.properties
```

```
# Kafka 브로커 주소 설정
```

```
bootstrap.servers=localhost:9092
```

Kafka Connect 클러스터 그룹 ID

group.id=connect-cluster

데이터 변환 설정 (JSON 사용)

key.converter=org.apache.kafka.connect.json.JsonConverter

value.converter=org.apache.kafka.connect.json.JsonConverter

key.converter.schemas.enable=false

value.converter.schemas.enable=false

내부 오프셋 저장용 Kafka 토픽 설정

offset.storage.topic=connect-offsets

offset.storage.replication.factor=1

내부 설정 저장용 Kafka 토픽 설정

config.storage.topic=connect-configs

config.storage.replication.factor=1

상태 저장용 Kafka 토픽 설정

status.storage.topic=connect-status

status.storage.replication.factor=1

플러그인(커넥터) 경로 설정

plugin.path=/opt/kafka/connect-plugins

REST API 리스너 활성화

listeners=HTTP://0.0.0.0:8083

로그 플러시 간격 설정

offset.flush.interval.ms=10000

sudo nano /opt/kafka/config/worker.properties

#/opt/kafka/config/worker.properties

bootstrap.servers=localhost:9092

group.id=connect-cluster

```
key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
key.converter.schemas.enable=true
value.converter.schemas.enable=true
offset.storage.topic=connect-offsets
offset.storage.replication.factor=1
config.storage.topic=connect-configs
config.storage.replication.factor=1
status.storage.topic=connect-status
status.storage.replication.factor=1
plugin.path=/opt/kafka/connect-plugins
```

2.3.11 JDBC Connector 설치

```
# Confluent Hub 클라이언트를 이용해 JDBC Connector 설치
confluent-hub install confluentinc/kafka-connect-jdbc:latest --component-
dir /opt/kafka/connect-plugins --worker-configs /opt/kafka/config/worker.
properties
```

```
# 또는 수동 다운로드
```

```
mkdir -p /opt/kafka/connect-plugins
cd /opt/kafka/connect-plugins
```

```
wget https://packages.confluent.io/maven/io/confluent/kafka-connect-jdb
c/10.8.2/kafka-connect-jdbc-10.8.2.jar
```

```
# /opt/kafka/connect-plugins 위치에서
```

```
# mysql connector 다운로드
```

```
wget https://repo1.maven.org/maven2/mysql/mysql-connector-java/8.2.0/
mysql-connector-java-8.2.0.jar
```

```
# mongo connector 다운로드
```

```
wget https://repo1.maven.org/maven2/org/mongodb/mongo-java-driver/3.1
2.14/mongo-java-driver-3.12.14.jar
```

2.3.12 Kafka Connect 서비스 등록

서비스 등록

```
sudo nano /etc/systemd/system/kafka-connect.service
```

```
# /etc/systemd/system/kafka-connect.service
```

```
[Unit]
```

```
Description=Kafka Connect Service
```

```
After=network.target kafka.service
```

```
[Service]
```

```
Type=simple
```

```
ExecStart=/opt/kafka/bin/connect-distributed.sh /opt/kafka/config/worker.  
properties
```

```
Restart=on-failure
```

```
User=ubuntu
```

```
Group=ubuntu
```

```
[Install]
```

```
WantedBy=multi-user.target
```

```
sudo systemctl daemon-reload
```

```
sudo systemctl enable kafka-connect
```

```
sudo systemctl start kafka-connect
```

정상 등록 확인

```
curl -X GET http://localhost:8083/connector-plugins
```

출력결과에 JDBC Connector 가 있으면 정상설치

```
[{"class":"io.confluent.connect.jdbc.JdbcSinkConnector","type":"sink","ver  
sion":"10.8.2"}, {"class":"io.confluent.connect.jdbc.JdbcSourceConnecto  
r","type":"source","version":"10.8.2"}, {"class":"org.apache.kafka.connect.  
mirror.MirrorCheckpointConnector","type":"source","version":"3.7.0"}, {"cla  
ss":"org.apache.kafka.connect.mirror.MirrorHeartbeatConnector","type":"s  
ource","version":"3.7.0"}, {"class":"org.apache.kafka.connect.mirror.MirrorS  
ourceConnector","type":"source","version":"3.7.0"}]
```


2.3.13 MySQL Source Connector 등록

```
cd /opt/kafka/config  
sudo nano mysql-source-connector.json
```

```
{  
  "name": "mysql-source-connector",  
  "config": {  
    "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",  
    "tasks.max": "1",  
    "connection.url": "jdbc:mysql://localhost:3306/{DB이름}", // keywi_feed  
    "connection.user": "{mysql 유저명}",  
    "connection.password": "{mysql 비밀번호}",  
    "mode": "timestamp+incrementing",  
    "timestamp.column.name": "timestamp",  
    "incrementing.column.name": "{ID COLUMN 이름}", // activity_id  
    "table.whitelist": "{등록할 테이블 이름}", // user_activities  
    "topic.prefix": "mysql-"  
  }  
}
```

```
# Source Connector 생성  
curl -X POST -H "Content-Type: application/json" --data @/opt/kafka/config/mysql-source-connector.json http://localhost:8083/connectors
```

2.3.14 MySQL Sink Connector 등록

```
sudo nano mysql-sink-connector.json
```

```
{  
  "name": "mysql-sink-connector",  
  "config": {  
    "connector.class": "io.confluent.connect.jdbc.JdbcSinkConnector",  
    "tasks.max": "1",  
    "connection.url": "jdbc:mysql://localhost:3306/{DB이름}", // keywi_feed  
    "connection.user": "{mysql 유저명}",  
  }  
}
```

```

"connection.password": "{mysql 비밀번호}",
"connection.pool.size": "5",
"batch.size": "3000",
"auto.create": "false",
"auto.evolve": "false",
"topics": "{카프카에서 등록된 토픽명/mysql-[테이블이름] }", // mysql-user-act
ivity-events
"table.name.format": "DB이름}.{테이블이름}", // keywi_feed.user_activities
"pk.mode": "record_value",
"pk.fields": "activity_id", // activity_id
"insert.mode": "upsert",
"errors.tolerance": "none",
"errors.log.enable": "true",
"errors.deadletterqueue.topic.name": "mysql-connector-errors"
}
}

```

Sink Connector 생성

```

curl -X POST -H "Content-Type: application/json" --data @/opt/kafka/config/mysql-sink-connector.json http://localhost:8083/connectors

```

2.3.15 Connector 등록 확인

Connector 목록 확인

```

curl -X GET http://localhost:8083/connectors

```

```

{"name":"mysql-source-connector","config":{"설정 내용"},"tasks":[],"type":"s
ource"}{"name":"mysql-sink-connector","config":{"설정 내용"},"tasks":[],"typ
e":"sink"}["mysql-source-connector","mysql-sink-connector"]

```

2.4 프론트 배포 서버 세팅

2.4.1 프론트 서버 방화벽 설정

```
sudo ufw allow OpenSSH
sudo ufw allow 80 # HTTP
sudo ufw allow 443 # HTTPS
sudo ufw enable
```

2.4.2 npm 설치

```
# nvm 다운로드 및 설치:
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.40.1/install.sh |
bash

# Node.js 다운로드 및 설치:
nvm install 22

# Node.js 버전 확인:
node -v # "v22.14.0"
nvm current # "v22.14.0"

npm 버전 확인:
npm -v # 10.9.2
```

2.4.3 프론트 NGINX 세팅

```
공통 세팅
...
    location / {
        root /var/www/keywi;
        index index.html;
        try_files $uri /index.html;
    }
    location = /50x.html {
        root /usr/share/nginx/html;
    }
}
```

3. Jenkins Pipeline

3.1 BE 배포 세팅

3.1.1 BE 빌드 후 배포

```
import groovy.json.JsonOutput

pipeline {
    agent any
    tools {
        maven 'Default Maven'
        jdk 'Zulu17'
        git 'Default Git'
    }
    environment {
        STAGE_NAME = ''

        // Docker 설정
        DOCKER_USER = 'team2room'
        IMAGE_NAME = 'keywi'
        GIT_COMMIT_SHORT = ''
        DOCKER_TAG = ''

        // 깃 정보
        COMMIT_MSG = ''
        COMMIT_HASH = ''
        AUTHOR = ''
        BRANCH_NAME = ''
        SERVICE_PATH = ''
        SERVICES = ''
        ERROR_MSG = "false"

        // 서버 정보
```

```

// TEST_SERVER = 'ssafy-gcloudtest.kro.kr'
PROD_SERVER = 'j12e202.p.ssafy.io'
JIRA_BASE_URL = 'https://ssafy.atlassian.net'
GITLAB_BASE_URL = 'https://lab.ssafy.com/s12-fintech-finance-sub1/
S12P21E202'
}
stages {
  stage('Checkout and Update') {
    steps {
      script {
        echo "Branch: ${env.GIT_BRANCH}"
        echo "Commit: ${env.GIT_COMMIT}"
        BRANCH_NAME = env.GIT_BRANCH.replaceFirst("refs/heads/",
""))

        STAGE_NAME = "Checkout and Update (1/6)"
        def repoExists = fileExists('.git')
        if (repoExists) {
          echo "Repository exists. Updating..."
          try {
            checkout([
              $class: 'GitSCM',
              branches: [[name: '*/${BRANCH_NAME}']],
              userRemoteConfigs: [[
                url: "${GITLAB_BASE_URL}.git",
                credentialsId: 'gitlab-credentials'
              ]],
              extensions: [
                [$class: 'CleanBeforeCheckout'],
                [$class: 'PruneStaleBranch']
              ]
            ])
            withCredentials([gitUsernamePassword(credentialsId: 'gitlab-credentials')]) {
              sh "git fetch --all --prune"
              sh "git checkout -B ${BRANCH_NAME} origin/${BRANCH_NAME} --force"
              sh "git pull origin ${BRANCH_NAME}"
            }
          }
        }
      }
    }
  }
}

```

```

        GIT_COMMIT_SHORT = sh(script: "git rev-parse --short
HEAD", returnStdout: true).trim()
        DOCKER_TAG = "${env.BUILD_NUMBER}-${GIT_COMMIT_SHORT}"

        // MSA 서비스 경로 설정
        SERVICE_PATH = BRANCH_NAME.contains('feature/B
E/') ? BRANCH_NAME.replace("feature/BE/", "") : ""

        // 서비스 목록 설정
        if (BRANCH_NAME == "master") {
            env.SERVICES = "config,eureka,gateway,auth,produc
t,feed,mypage,financial,board,chat,search"
        } else if (BRANCH_NAME == "feature/BE/gateway") {
            SERVICES = "eureka,gateway"
        } else if (BRANCH_NAME == "feature/BE/pay") {
            SERVICES = "financial"
        } else {
            SERVICES = "${SERVICE_PATH}"
        }

        echo "branch: ${BRANCH_NAME}"
        echo "services to build: ${SERVICES}"
        echo "docker tag: ${DOCKER_TAG}"
    }
} catch (Exception e) {
    echo "Error during update: ${e.message}"
    ERROR_MSG = "Failed to update repository"
    error ERROR_MSG
}
} else {
    echo "Repository does not exist. Cloning..."
    try {
        withCredentials([gitUsernamePassword(credentialsId: 'gitl
ab-credentials')]) {
            sh "git clone ${GITLAB_BASE_URL}.git ."

            sh "git checkout ${BRANCH_NAME}"

```

```

        GIT_COMMIT_SHORT = sh(script: "git rev-parse --short
HEAD", returnStdout: true).trim()
        DOCKER_TAG = "${env.BUILD_NUMBER}-${GIT_COMMIT_SHORT}"

        // MSA 서비스 경로 설정
        SERVICE_PATH = BRANCH_NAME.replace("feature/B
E/", "")

        // 서비스 목록 설정
        if (BRANCH_NAME == "master") {
            env.SERVICES = "config,eureka,gateway,auth,product,feed,mypage,financial,board,chat,search"
        } else if (BRANCH_NAME == "feature/BE/gateway") {
            SERVICES = "eureka,gateway"
        } else if (BRANCH_NAME == "feature/BE/pay") {
            SERVICES = "financial"
        } else {
            SERVICES = "${SERVICE_PATH}"
        }

        echo "branch: ${BRANCH_NAME}"
        echo "services to build: ${SERVICES}"
        echo "docker tag: ${DOCKER_TAG}"
    }
} catch (Exception e) {
    echo "Error during clone: ${e.message}"
    ERROR_MSG = "Failed to clone repository"
    error ERROR_MSG
}
}

AUTHOR = sh(script: "git log -1 --pretty=format:%an", returnStdout: true).trim()
COMMIT_MSG = sh(script: 'git log -1 --pretty=%B', returnStdout: true).trim()
COMMIT_HASH = sh(script: "git log -1 --pretty=format:%H", returnStdout: true).trim()

```

```

    }
    script {
        if (sh(
            script: "git ls-tree -d origin/${BRANCH_NAME} BE",
            returnStatus: true
        ) != 0) {
            currentBuild.result = 'ABORTED'
            ERROR_MSG = "BE 디렉토리가 원격 브랜치에 존재하지 않음"
            error ERROR_MSG
        }

        if (!fileExists("BE")) {
            ERROR_MSG = "BE 디렉토리가 로컬에 존재하지 않음"
            error ERROR_MSG
        }

        if (COMMIT_MSG.toLowerCase().contains('[fe]')) {
            ERROR_MSG = "FE 커밋으로 빌드 중단"
            error ERROR_MSG
        }

        if (COMMIT_MSG.toLowerCase().contains('merge') && COMMIT_MSG.toLowerCase().contains('feature/fe')) {
            ERROR_MSG = "FE 커밋으로 빌드 중단"
            error ERROR_MSG
        }
    }
}

stage('Inject Config') {
    steps {
        script {
            def servicesList = SERVICES.split(',')
            STAGE_NAME = "Inject Config (2/6)"

            servicesList.each { SERVICE →
                echo "Config Searching..."
            }
        }
    }
}

```



```

        if (SERVICE == "config") {
            echo "Injecting config for ${SERVICE}..."
            withCredentials([
                file(credentialsId: 'config_yml', variable: 'CONFIG_FILE')
            ]) {
                sh """
                    mkdir -p BE/${SERVICE}/src/main/resources
                    cp \${CONFIG_FILE} BE/${SERVICE}/src/main/resource
s/application.yml
                """
            }
        }
    }
}

stage('Build') {
    steps {
        script {
            STAGE_NAME = "Build (3/6)"
            def servicesList = SERVICES.split(',')
            servicesList.each { SERVICE →
                echo "Building ${SERVICE}..."
                dir("BE/${SERVICE}") {
                    try {
                        // Maven 빌드 사용
                        sh "mvn clean package -DskipTests"
                    } catch (Exception e) {
                        ERROR_MSG = e.getMessage()
                        error "Build failed for ${SERVICE}: ${ERROR_MSG}"
                    }
                }
            }
        }
    }
}

post {
    failure {
        cleanWs()
    }
}

```

```

        script {
            ERROR_MSG += "\nBuild failed"
            error ERROR_MSG
        }
    }
}
}
stage('Docker Build') {
    steps {
        script {
            STAGE_NAME = "Docker Build (4/6)"
            def servicesList = SERVICES.split(',')
            servicesList.each { SERVICE →
                echo "Docker building ${SERVICE}..."
                def jarFile = sh(script: "ls BE/${SERVICE}/target/*.jar", return
Stdout: true).trim()

                try {
                    // docker.build("${DOCKER_USER}/${IMAGE_NAME}-${SE
RVICE}:${DOCKER_TAG}-test",
                    //    "--no-cache --build-arg JAR_FILE=${jarFile} -f BE/
${SERVICE}/Dockerfile .")

                    docker.build("${DOCKER_USER}/${IMAGE_NAME}-${SER
VICE}:${DOCKER_TAG}",
                        "--no-cache --build-arg JAR_FILE=${jarFile} -f BE/${SE
RVICE}/Dockerfile .")
                    sh "docker tag ${DOCKER_USER}/${IMAGE_NAME}-${SER
VICE}:${DOCKER_TAG} ${DOCKER_USER}/${IMAGE_NAME}-${SERVICE}:la
test"

                    // sh "docker save ${DOCKER_USER}/${IMAGE_NAME}
-${SERVICE}:${DOCKER_TAG}-test | gzip > ${SERVICE}-image.tar.gz"
                } catch (Exception e) {
                    ERROR_MSG = e.getMessage()
                    error "Docker build failed for ${SERVICE}: ${ERROR_MS
G}"
                }
            }
        }
    }
}

```

```

    }
  }
  post {
    failure {
      cleanWs()
      script {
        ERROR_MSG += "\nDocker Build failed"
        error ERROR_MSG
      }
    }
  }
}

stage('Push to Docker Hub') {
  steps {
    script {
      STAGE_NAME = "Push to Docker Hub (5/6)"/"/"Deploy to Test
(5/9)"

      def servicesList = SERVICES.split(',')
      servicesList.each { SERVICE →
        echo "Pushing ${SERVICE} to Docker Hub..."

        docker.withRegistry('https://index.docker.io/v1/', 'keywi-dock
er') {
          docker.image("${DOCKER_USER}/${IMAGE_NAME}-${S
ERVICE}:${DOCKER_TAG}").push()
        }
        echo "Push ${SERVICE} Docker Image."
      }
    }
  }
  post {
    failure {
      cleanWs()
      script {
        ERROR_MSG = "Docker Push failed"
        error ERROR_MSG
      }
    }
  }
}

```

```

    }
  }
  stage('Deploy to Prod') {
    steps {
      script {
        STAGE_NAME = "Deploy to Prod (6/6)"
        def servicesList = SERVICES.split(',')
        servicesList.each { SERVICE →
          echo "Deploying ${SERVICE} to production server..."
          def memoryl=""
          if (SERVICE == 'search'){
            memoryl = " --memory=1.3g --memory-swap=1.3g"
          } else if (SERVICE == 'config'){
            memoryl = " --memory=512m --memory-swap=512m"
          } else if (SERVICE == 'chat'){
            memoryl = " --memory=2g --memory-swap=2g"
          } else if (SERVICE == 'auth' || SERVICE == 'feed'){
            memoryl = " --memory=1g --memory-swap=1g"
          } else {
            memoryl = " --memory=768m --memory-swap=1g"
          }
          sshagent(['ec2-ssafy']) {
            sh """
              ssh -o StrictHostKeyChecking=no ubuntu@${PROD_SERVER} "
                docker pull ${DOCKER_USER}/${IMAGE_NAME}-${SERVICE}:${DOCKER_TAG}
                docker stop ${SERVICE} || true
                docker rm ${SERVICE} || true
                docker run${memoryl} -d --network host --name ${SERVICE} ${DOCKER_USER}/${IMAGE_NAME}-${SERVICE}:${DOCKER_TAG}
              "
            """
          }
        }
        echo "Success Production deployment."
      }
    }
  }
}

```

```

    }
    post {
        failure {
            script {
                ERROR_MSG = "Production deployment failed"
                error ERROR_MSG
            }
        }
    }
}
stage('Complete') {
    steps {
        script {
            STAGE_NAME = "Completed"
        }
    }
}
}
post {
    always {
        script {
            def issueKeyPattern = /[#{S12P21E202-\d+}]/
            def issueKey = (COMMIT_MSG =~ /S12P21E202-\d+/) ? (COMMIT_MSG =~ /S12P21E202-\d+/)[0] : null
            def cleanedMessage = issueKey ? COMMIT_MSG.replaceFirst(issueKeyPattern, '').trim() : COMMIT_MSG
            def jiraLink = issueKey ? "${JIRA_BASE_URL}/jira/software/c/projects/S12P21E202/boards/7980?selectedIssue=${issueKey}" : ""

            def message = "${env.JOB_NAME} - #${env.BUILD_NUMBER}\n"
            + "- 결과: " +
                (cleanedMessage.toLowerCase().contains('[fe]') ? "STOP\n" : "${currentBuild.currentResult}\n") +
                "- 브랜치: ${BRANCH_NAME}\n- 서비스: ${SERVICES}\n- 커밋: " +
                (issueKey ? "[${issueKey}] " : "") +
                "[${cleanedMessage}](${GITLAB_BASE_URL}/-/commit/${COMMIT_HASH}) (${GIT_COMMIT_SHORT}) [${AUTHOR}]\n" +

```

```

        "- 실행 시간: ${currentBuild.durationString}\n" +
        "- 최종 실행된 스테이지 : ${STAGE_NAME}\n" +
        ((ERROR_MSG!="false") ? "- ERROR :\n`${ERROR_MSG}`\n" : "")

    if (issueKey) {
        try {
            def requestBody = [body: message]
            def response = httpRequest authentication: 'jira-credentials',
            contentType: 'APPLICATION_JSON',
            httpMode: 'POST',
            requestBody: groovy.json.JsonOutput.toJson(requestBody),
            url: "${JIRA_BASE_URL}/rest/api/2/issue/${issueKey}/comment"

            echo "JIRA comment added successfully. Status: ${response.status}"
        } catch(e) {
            echo "JIRA 코멘트 추가 실패: ${e.message}"
        }
        message += (currentBuild.currentResult == 'ABORTED' ? "- **사용자 취소**\n" : "")
        message += "- 상세: " + (currentBuild.currentResult == 'SUCCESS' ? ":jenkins7:" : (currentBuild.currentResult == 'ABORTED' ? ":jenkins_cute_flip:" : (cleanedMessage.toLowerCase().contains('[fe]') ? ":jenkins1:" : ":jenkins5:"))) + " [Jenkins](${env.BUILD_URL})"
        message += jiraLink ? " | jira: [Jira](${jiraLink}) " : (cleanedMessage.contains('Merge') ? " | jira6: [Jira](${JIRA_BASE_URL}/jira/software/c/projects/S12P21E202/boards/7980)" : " | jira3:")
        message += "\n\n`${env.BUILD_TIMESTAMP}`"

        mattermostSend color: currentBuild.currentResult == 'SUCCESS' ? 'good' : (cleanedMessage.toLowerCase().contains('[fe]') ? 'good' : (currentBuild.currentResult == 'ABORTED' ? 'warning' : 'danger')), message: message
    }
}
script {

```

```

        cleanWs(cleanWhenNotBuilt: false,
            deleteDirs: true,
            disableDeferredWipeout: true,
            notFailBuild: true)
    }
}
failure {
    script {
        cleanWs(cleanWhenNotBuilt: false,
            deleteDirs: true,
            disableDeferredWipeout: true,
            notFailBuild: true)
    }
}
}
}
}

```

3.1.2 Spring Cloud Config 갱신

```

pipeline {
    agent any
    tools {
        git 'Default Git'
    }
    environment {
        GITHUB_URL = 'https://github.com/PoloCeleste/KeyWi_Config'
        SERVER_TO_UPDATE = ''
    }
    stages {
        stage('Clone Mirror') {
            steps {
                script {
                    def repoExists = fileExists('.git')
                    if (repoExists) {
                        echo "Repository exists. Updating..."
                        try {
                            checkout([
                                $class: 'GitSCM',

```

```

        branches: [[name: '*/master']],
        userRemoteConfigs: [[
            url: "${GITHUB_URL}.git",
            credentialsId: 'github_credentials'
        ]],
        extensions: [
            [$class: 'CleanBeforeCheckout'],
            [$class: 'PruneStaleBranch']
        ]
    ])
    withCredentials([gitUsernamePassword(credentialsId: 'github_credentials')]) {
        sh """
            git fetch --all --prune
            git remote update --prune
            git gc --prune=now
        """

        sh "git checkout master"
        sh "git pull -f"
    }
} catch (Exception e) {
    echo "Error during update: ${e.message}"
    error "Failed to update repository"
}
} else {
    echo "Repository does not exist. Cloning..."
    try {
        withCredentials([gitUsernamePassword(credentialsId: 'github_credentials')]) {
            sh "git clone ${GITHUB_URL}.git ."
            sh "git pull"
        }
    } catch (Exception e) {
        echo "Error during clone: ${e.message}"
        error "Failed to clone repository"
    }
}
}

```



```

        echo "Branch mirror Successfull."
    }
}
}
stage('Detect Change Server') {
    steps {
        script {
            def changedFiles = sh(script: "git diff --name-only HEAD~1 HEAD", returnStdout: true).trim()
            def serversToUpdate = []

            changedFiles.split('\n').each { file →
                serversToUpdate.add(file.split('/')[0])
            }

            env.SERVERS_TO_UPDATE = serversToUpdate.join(',')
        }
    }
}
stage('Send Refresh') {
    steps {
        script {
            SERVERS_TO_UPDATE.split(',').each { server →
                sh "curl -X POST https://j12e202.p.ssafy.io/config/actuator/b
usrefresh/${server}"
            }
        }
    }
}
post {
    always {
        script {
            cleanWs(cleanWhenNotBuilt: false,
                deleteDirs: true,
                disableDeferredWipeout: true,
                notFailBuild: true)
        }
    }
}

```

```
}  
}  
}
```

3.2 FE 배포 세팅

```
pipeline {  
  agent any  
  tools {  
    git 'Default Git'  
  }  
  environment {  
    GITLAB_BASE_URL = 'https://lab.ssafy.com/s12-fintech-finance-sub1/  
S12P21E202'  
    BRANCH_NAME = 'develop'  
    SERVER_USER = 'celeste'  
    SERVER_HOST = 'keywi.polocelste.site'  
    SERVER_WORK_DIR = '/home/celeste/KeyWi/FE/keywi'  
  }  
  stages {  
    stage('Clone Mirror') {  
      steps {  
        script {  
          def repoExists = fileExists('.git')  
          if (repoExists) {  
            echo "Repository exists. Updating..."  
            try {  
              checkout([  
                $class: 'GitSCM',  
                branches: [[name: '*/develop']],  
                userRemoteConfigs: [[  
                  url: "${GITLAB_BASE_URL}.git",  
                  credentialsId: 'gitlab-credentials'  
                ]],  
                extensions: [  
                  [$class: 'CleanBeforeCheckout'],  

```

```

        [$class: 'PruneStaleBranch']
    ]
})
withCredentials([gitUsernamePassword(credentialsId: 'gitlab-credentials')]) {
    sh """
        git fetch --all --prune
        git remote update --prune
        git gc --prune=now
    """

    sh "git checkout -B ${BRANCH_NAME} origin/${BRANCH_NAME} --force"

    sh '''
        git fetch --all

        git branch -r | grep -v '\\->' | while read remote; do
            local_branch="${remote#origin/}"

            if ! git rev-parse --verify "$local_branch" >/dev/null
2>&1; then
                git branch --track "$local_branch" "$remote"
            fi
        done

        git pull --all
    '''
}
} catch (Exception e) {
    echo "Error during update: ${e.message}"
    error "Failed to update repository"
}
} else {
    echo "Repository does not exist. Cloning..."
    try {
        withCredentials([gitUsernamePassword(credentialsId: 'gitlab-credentials')]) {

```

```

        sh "git clone ${GITLAB_BASE_URL}.git ."
        sh "git checkout develop"

        sh '''
            git fetch --all

            git branch -r | grep -v '\\->' | while read remote; do
                local_branch="${remote#origin/}"

                if ! git rev-parse --verify "$local_branch" >/dev/null
2>&1; then
                    git branch --track "$local_branch" "$remote"
                fi
            done

            git pull --all
        '''
    }
} catch (Exception e) {
    echo "Error during clone: ${e.message}"
    error "Failed to clone repository"
}
}
echo "Branch mirror Successfull."
}
}
}
stage('Push Mirror') {
    steps {
        script {
            withCredentials([gitUsernamePassword(credentialsId: 'github_c
redentials')]) {
                sh """
                    git push -f --mirror https://github.com/PoloCeleste/KeyWi.
git
                    """
            }
        }
    }
}

```

```

        echo "Mirror Push Successfull."
    }
}
}
stage('Clone and Checkout') {
    steps {
        script {
            withCredentials([usernamePassword(credentialsId: 'gitlab-credentials', usernameVariable: 'GITLAB_USER', passwordVariable: 'GITLAB_PASSWORD')]) {
                sshagent(['ody-server']) {
                    sh """
                        ssh -o StrictHostKeyChecking=no ${SERVER_USER}@${SERVER_HOST} "
                            if [ ! -d ~/KeyWi ]; then // 홈에 키위폴더 없으면
                                echo 'KeyWi folder does not exist'
                                cd ~ && // 클론 받아서 체크아웃
                                git clone https://${GITLAB_USER}:${GITLAB_PASSWORD}@lab.ssafy.com/s12-fintech-finance-sub1/S12P21E202.git KeyWi &&
                                cd KeyWi &&
                                git checkout -B ${BRANCH_NAME} origin/${BRANCH_NAME} --force
                            else // 존재하면
                                echo 'KeyWi folder exists'
                                cd ~/KeyWi && // 키위폴더 디벨롭 브랜치로 갱신
                                git fetch --all &&
                                git reset --hard origin/${BRANCH_NAME}
                            fi
                        "
                    """
                }
            }
            echo "Branch checked out Successfull"
        }
    }
}
stage('Send .env') {
    steps {

```

```

    script {
      withCredentials([file(credentialsId: 'react-env', variable: 'ENV_FILE')]) {
        sshagent(['ody-server']) {
          sh """
            ssh -o StrictHostKeyChecking=no ${SERVER_USER}@
            ${SERVER_HOST} "
              cd ${SERVER_WORK_DIR}
              rm .env // 기존꺼 지우고 -> 권한문제
            "
            scp -o StrictHostKeyChecking=no "$ENV_FILE" ${SERVER_USER}@${SERVER_HOST}:${SERVER_WORK_DIR}/.env // 새로 전송
          """
        }
      }

      echo ".env sent."
    }
  }

  stage('Build on Remote Server') {
    steps {
      script {
        sshagent(['ody-server']) {
          sh """
            ssh -o StrictHostKeyChecking=no ${SERVER_USER}@${SERVER_HOST} "
              curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash // nvm 설치
              source ~/.zshrc
              cd ${SERVER_WORK_DIR}
              nvm use --lts // 최신버전 사용 -> 최신버전 설치되어 있으나,
              계속 오래된 버전이 잡힘. 직접 ssh로 접속해서 해보면 최신버전 잘 잡힘
              npm -v
              node -v
              npm install // 패키지 설치하고
              npm run build // 페이지 빌드 후
              sudo rm -rf /var/www/keywi
            "
          """
        }
      }
    }
  }
}

```

```

        sudo mv /home/celeste/KeyWi/FE/keywi/dist /var/www/
keywi // 배포 위치로 이동
        sudo systemctl restart nginx // 엔지닉스 재시작
        "
        ""
    }
    echo "Built Completed."
}
}
}
}
}
post {
    always {
        script {
            def message = "${env.JOB_NAME} - #${env.BUILD_NUMBER}\n"
+ "- 결과: " +
                "${currentBuild.currentResult}\n" +
                "- 브랜치: ${BRANCH_NAME}\n- 서비스: 프론트 배포\n" +
                "- 실행 시간: ${currentBuild.durationString}\n" +
                "### 배포 로직 실행 완료.\n"
            message += "\n\n`${env.BUILD_TIMESTAMP}`"

            mattermostSend color: currentBuild.currentResult == 'SUCCESS'
? 'good' : (currentBuild.currentResult == 'ABORTED' ? 'warning' : 'danger'),
message: message
        }
        echo "Cleaning."
        cleanWs(cleanWhenNotBuilt: false,
            deleteDirs: true,
            disableDeferredWipeout: true,
            notFailBuild: true)
    }
}
}
}

```

4. Front 세팅

*npm 패키지 이름 규칙

1. 소문자만 사용 가능
2. 공백, 특수 문자 금지
3. 대시(-), 밑줄(_), 숫자 사용 가능
4. 예약어 사용 금지 (test, npm, react 같은 예약어 ❌)

```
npm create vite@latest keywi --template react-ts
```

1 vite + react + ts 설치

```
npm create vite@latest keywi --template react-ts  
→ React, TypeScript 선택
```

```
cd 폴더
```

```
npm install
```

2 pwa 설치

```
npm install vite-plugin-pwa
```

3 tailwindcss + styled-components + twin.macro 설치

```
# v3 버전 설치 방법  
# Tailwind CSS v3.x 설치  
npm i -D tailwindcss@3 postcss autoprefixer  
npx tailwind init -p  
  
# styled-components 설치 (TypeScript 지원 포함)  
npm install styled-components
```



```
npm install -D @types/styled-components
```

```
# twin.macro 설치
```

```
npm install twin.macro
```

```
npm install vite-plugin-babel-macros
```

```
npm install -D @emotion/react @emotion/styled
```

파일 수정

public 폴더 아래 → icons 폴더 생성 → 아래 파일들 넣기

[favicomatic.zip](#)

```
// vite.config.ts
```

```
import { defineConfig } from 'vite'
```

```
import react from '@vitejs/plugin-react'
```

```
import { VitePWA } from 'vite-plugin-pwa';
```

```
import macrosPlugin from 'vite-plugin-babel-macros'
```

```
// https://vite.dev/config/
```

```
export default defineConfig({
```

```
  plugins: [
```

```
    react(),
```

```
    VitePWA({
```

```
      registerType: 'autoUpdate',
```

```
      devOptions: {
```

```
        enabled: true, // 개발환경에서 pwa 기능활성화
```

```
    },
```

```
    manifest: {
```

```
      name: 'keywi',
```

```
      short_name: '키위',
```

```
      description: '1:1 키보드 견적 서비스, 나만의 키보드를 맞추고 뽐내보세요.',
```

```
      start_url: '/',
```

```
      display: 'standalone', // 네이티브앱처럼 화면 전체를 채움
```

```
background_color: '#ffffff',
theme_color: '#ffffff',
lang: 'ko',
"icons": [
  {
    "src": "icons/apple-touch-icon-57×57.png",
    "sizes": "57×57",
    "type": "image/png"
  },
  {
    "src": "icons/apple-touch-icon-60×60.png",
    "sizes": "60×60",
    "type": "image/png"
  },
  {
    "src": "icons/apple-touch-icon-72×72.png",
    "sizes": "72×72",
    "type": "image/png"
  },
  {
    "src": "icons/apple-touch-icon-114×114.png",
    "sizes": "114×114",
    "type": "image/png"
  },
  {
    "src": "icons/apple-touch-icon-120×120.png",
    "sizes": "120×120",
    "type": "image/png"
  },
  {
    "src": "icons/apple-touch-icon-144×144.png",
    "sizes": "144×144",
    "type": "image/png"
  },
  {
    "src": "icons/apple-touch-icon-152×152.png",
    "sizes": "152×152",
    "type": "image/png"
  }
]
```

```

    },
    {
      "src": "icons/apple-touch-icon-180×180.png",
      "sizes": "180×180",
      "type": "image/png"
    },
    {
      "src": "icons/favicon-32×32.png",
      "sizes": "32×32",
      "type": "image/png"
    },
    {
      "src": "icons/favicon-96×96.png",
      "sizes": "96×96",
      "type": "image/png"
    },
    {
      "src": "icons/favicon-16×16.png",
      "sizes": "16×16",
      "type": "image/png"
    },
    {
      "src": "icons/logo192.png",
      "sizes": "192×192",
      "type": "image/png"
    },
    {
      "src": "icons/logo512.png",
      "sizes": "512×512",
      "type": "image/png"
    }
  ],
},
}),
macrosPlugin(),
],
})

```

```
// tailwind.config.js 수정

/** @type {import('tailwindcss').Config} */
export default {
  content: [
    "./index.html",
    "./src/**/*.js,ts,jsx,tsx",
  ],
  theme: {
    extend: {},
  },
  plugins: [],
}
```

```
// postcss.config.js 수정

export default {
  plugins: {
    'postcss-import': {},
    tailwindcss: {},
    autoprefixer: {},
  },
}
```

4 zustand 설치

```
npm install zustand
```

5 TanStack Query 설치

```
npm install @tanstack/react-query
```

6 prettier, eslint 설치

1. Prettier와 ESLint 관련 플러그인을 설치

```
npm install --save-dev prettier eslint-config-prettier eslint-plugin-prettier
```

설치된 패키지 설명:

- **prettier**: 코드 포매팅 도구
- **eslint-config-prettier**: ESLint와 Prettier 간의 충돌을 방지
- **eslint-plugin-prettier**: ESLint에서 Prettier 규칙을 실행

2. **eslint.config.js** 수정

```
import js from '@eslint/js'
import globals from 'globals'
import reactHooks from 'eslint-plugin-react-hooks'
import reactRefresh from 'eslint-plugin-react-refresh'
import tseslint from 'typescript-eslint'
import prettierConfig from 'eslint-config-prettier'
import prettierPlugin from 'eslint-plugin-prettier'

export default tseslint.config(
  { ignores: ['**/*', '!src', '!src/'] }, // .prettiignore 과 동일하게 설정
  {
    extends: [
      js.configs.recommended,
      ...tseslint.configs.recommended,
      prettierConfig, // Prettier 설정 추가 (충돌 방지)
    ],
    files: ['**/*.ts', '**/*.tsx'],
    languageOptions: {
      ecmaVersion: 2020,
      globals: globals.browser,
    },
    plugins: {
      'react-hooks': reactHooks,
      'react-refresh': reactRefresh,
      prettier: prettierPlugin, // Prettier 플러그인 추가
    },
  },
)
```

```

rules: {
  ...reactHooks.configs.recommended.rules,
  'react-refresh/only-export-components': [
    'warn',
    { allowConstantExport: true },
  ],
  'prettier/prettier': 'error', // Prettier 규칙을 ESLint에서 에러로 표시
},
},
)

```

2. `.prettierignore` 파일 생성

```

**

!src/
!src/**

```

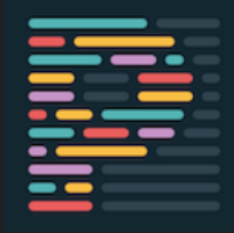
3. `.prettierrc` 파일 생성


```


{
  "semi": false,
  "singleQuote": true,
  "trailingComma": "all",
  "tabWidth": 2,
  "printWidth": 80,
  "arrowParens": "always",
  "endOfLine": "auto",
  "bracketSpacing": true
}


```

4. VS code extension 에서 아래 설치



Prettier - Code formatter
 Prettier  prettier.io | 55,252,877 | ★★★★★ (476) | ❤️ 스폰서
 Code formatter using prettier
 사용 안 함 ▼ 제거 ▼ ☒ 자동 업데이트 ⚙️



ESLint
 Microsoft  microsoft.com | 41,716,748 | ★★★★★ (241)
 Integrates ESLint JavaScript into VS Code.
 사용 안 함 ▼ 제거 ▼ 시험판 버전으로 전환 ☒ 자동 업데이트 ⚙️

5. VS Code 에서 ctrl + shift + p → settings.json 에 추가

```
// settings.json

"editor.defaultFormatter": "esbenp.prettier-vscode",
"editor.formatOnSave": true,
"editor.codeActionsOnSave": {
  "source.fixAll.eslint": "explicit"
},
```

6. ctrl + s 눌러서 저장시키면 변환되는 것을 볼 수 있음

7 ngrok 설치 - 로컬에서 http → https 변환용 (배포 전)



npm을 이용한 간단한 설치 방법

폴더 → `npm install -g ngrok` → 버전 확인 `ngrok -v` → 실행 `ngrok http 포트번호`

<https://ngrok.com/downloads/windows?tab=download>

다운받은 파일의 압축을 해제하고 ngrok.exe를 관리자 권한으로 실행

1. Ngrok 계정 생성

1. [Ngrok 공식 웹사이트](#)로 이동
2. 계정을 만들고 로그인

2. 인증 토큰 확인

1. 로그인 후 [Ngrok 인증 토큰 페이지](#)로 이동
2. `authtoken` 을 복사

3. 로컬 환경에 Ngrok 인증 등록

터미널에서 다음 명령어 실행 (복사한 `authtoken` 을 붙여넣기)

```
ngrok config add-authtoken <YOUR_AUTH_TOKEN>
```

4. Ngrok 다시 실행

```
ngrok http 3000 # 예제 (3000번 포트 노출)
```

*기존 인증 토큰 재설정

만약 이전에 등록한 인증 토큰이 잘못되었거나 만료된 경우, 아래 명령어로 다시 설정

```
ngrok authtoken <NEW_AUTH_TOKEN>
```

5. `vite.config.js` 수정

`vite.config.js` 파일을 열고, `server.allowedHosts` 옵션을 추가하여 ngrok 도메인을 허용합니다.

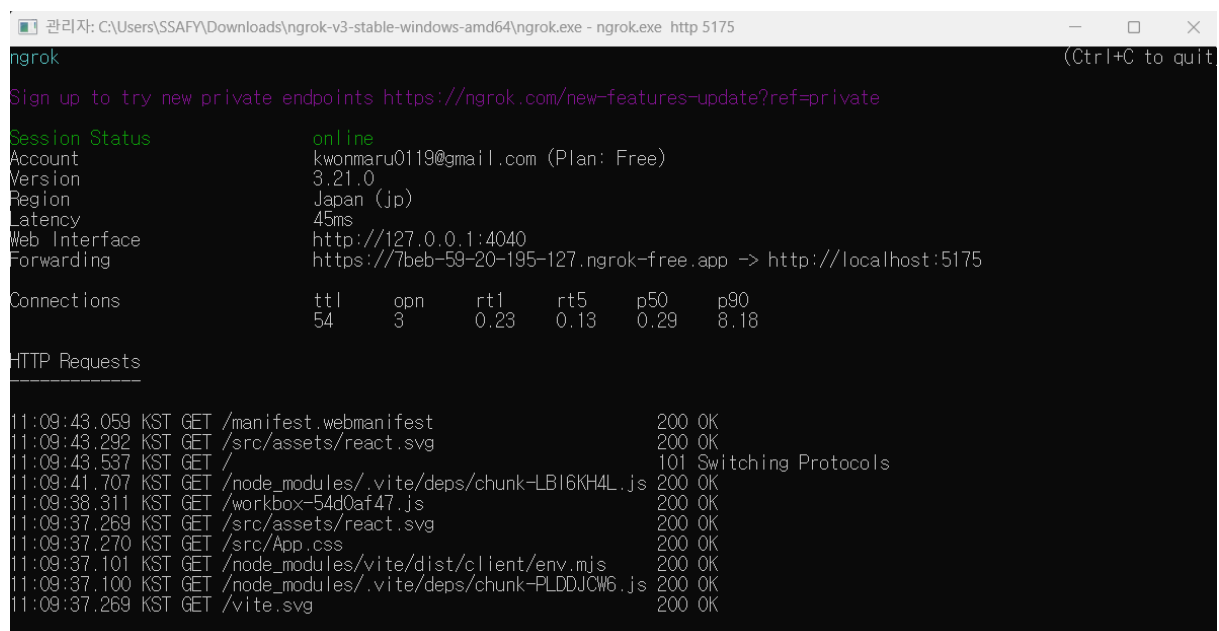
```
javascriptimport { defineConfig } from 'vite'

export default defineConfig({
  server: {
    allowedHosts: ['2623-59-20-195-127.ngrok-free.app'], // ngrok 주소를 명
    시적으로 추가
  },
})
```


만약 ngrok 주소가 매번 바뀌어 관리가 번거롭다면, 모든 호스트를 허용하도록 설정할 수도 있습니다. 하지만 이는 보안상 권장되지 않습니다.

```
javascriptexport default defineConfig({
  server: {
    allowedHosts: true, // 모든 호스트를 허용 (DNS 리바인딩 공격에 취약할 수 있음)
  },
})
```

Forwarding 주소로 접근



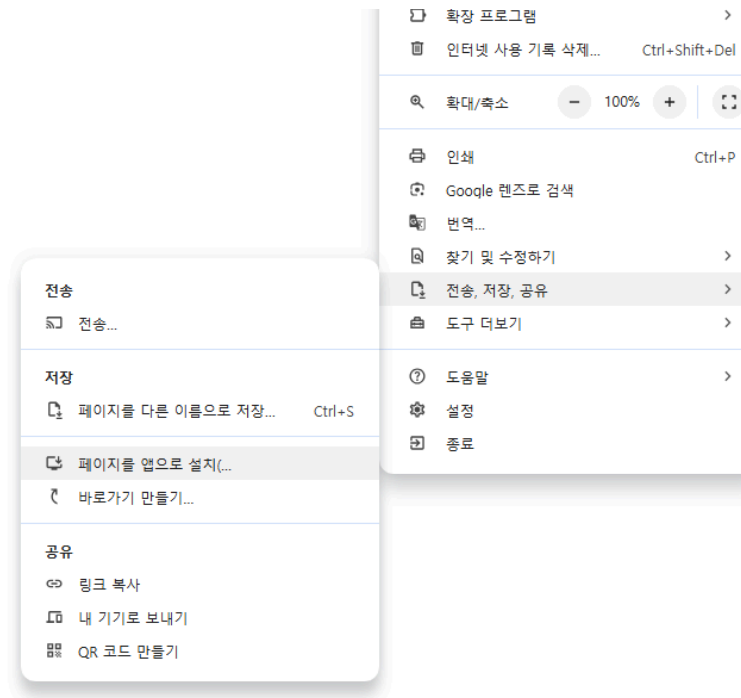
The screenshot shows the ngrok terminal window. The title bar indicates the path to the ngrok.exe file and the current URL (http 5175). The terminal output includes a sign-up link, session status (online), account information (kwonmaru0119@gmail.com), version (3.21.0), region (Japan), latency (45ms), web interface (http://127.0.0.1:4040), and forwarding URL (https://7beb-59-20-195-127.ngrok-free.app -> http://localhost:5175). Below this, a table shows connection statistics (t1, opn, rt1, rt5, p50, p90). At the bottom, a list of HTTP requests is shown with timestamps, methods, paths, and status codes.

```
관리자: C:\Users\SSAFY\Downloads\ngrok-v3-stable-windows-amd64\ngrok.exe - ngrok.exe http 5175
ngrok
Sign up to try new private endpoints https://ngrok.com/new-features-update?ref=private
Session Status      online
Account             kwonmaru0119@gmail.com (Plan: Free)
Version             3.21.0
Region              Japan (jp)
Latency             45ms
Web Interface       http://127.0.0.1:4040
Forwarding           https://7beb-59-20-195-127.ngrok-free.app -> http://localhost:5175

Connections          t1      opn      rt1      rt5      p50      p90
                   54       3       0.23     0.13     0.29     8.18

HTTP Requests
-----
11:09:43.059 KST GET /manifest.webmanifest 200 OK
11:09:43.292 KST GET /src/assets/react.svg 200 OK
11:09:43.537 KST GET / 101 Switching Protocols
11:09:41.707 KST GET /node_modules/.vite/deps/chunk-LB16KH4L.js 200 OK
11:09:38.311 KST GET /workbox-54d0af47.js 200 OK
11:09:37.269 KST GET /src/assets/react.svg 200 OK
11:09:37.270 KST GET /src/App.css 200 OK
11:09:37.101 KST GET /node_modules/vite/dist/client/env.mjs 200 OK
11:09:37.100 KST GET /node_modules/.vite/deps/chunk-PLDDJCW6.js 200 OK
11:09:37.269 KST GET /vite.svg 200 OK
```

***크롬에서 주소창에 앱 설치 아이콘이 안 보일 때**



8 실행

```
npm run dev
```

```
npm run build
```

```
npm run preview
```

9 절대 경로 설정

```
npm install --save-dev @types/node
```

1. vite.config.ts

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import path from 'path';

export default defineConfig({
  plugins: [react()],
  resolve: {
    alias: [
      { find: '@', replacement: path.resolve(__dirname, 'src') }
    ]
  }
})
```

```
],  
}  
});
```

2. tsconfig.json

```
{  
  "files": [],  
  "references": [  
    { "path": "./tsconfig.node.json" },  
    { "path": "./tsconfig.app.json" },  
  ],  
  "compilerOptions": {  
    "baseUrl": ".", // 절대 경로의 기준을 프로젝트 루트로 설정  
    "paths": {  
      "@/*": ["src/*"] // '@'를 'src' 폴더로 매핑  
    }  
  }  
}
```

3. tsconfig.app.json

```
{  
  "compilerOptions": {  
    "target": "ES2020",  
    "useDefineForClassFields": true,  
    "lib": ["ES2020", "DOM", "DOM.Iterable"],  
    "module": "ESNext",  
    "skipLibCheck": true,  
  
    /* Bundler mode */  
    "moduleResolution": "bundler",  
    "allowImportingTsExtensions": true,  
    "isolatedModules": true,  
    "moduleDetection": "force",  
    "noEmit": true,  
    "jsx": "react-jsx",  
  }  
}
```

```

/* Linting */
"strict": true,
"noUnusedLocals": true,
"noUnusedParameters": true,
"noFallthroughCasesInSwitch": true,

/* Path */
"baseUrl": ".", // 절대 경로의 기준을 프로젝트 루트로 설정
"paths": {
  "@/*": ["src/*"] // '@'를 'src' 폴더로 매핑
},
"include": ["src"]
}

```

10 라우터 설치

```
npm install react-router-dom
```

1 1 Shadcn 설치 (UI 라이브러리)

```
npx shadcn@latest init
```

```
newyork - neutral
```

(사용할 UI 설치)

```
npx shadcn@latest add drawer
```

1 2 axios 설치

```
npm i axios
```

1 3 react cookie 설치

```
npm install react-cookie
```

```
// .env
VITE_KAKAO_REST_API_KEY={카카오 API 키}
VITE_KAKAO_REDIRECT_URI={카카오에 등록된 프론트 배포 주소}/callback/kak
ao
VITE_BASE_URL={백엔드 배포 주소}
```

4. 엘라스틱 서치 세팅

```
docker pull docker.elastic.co/elasticsearch/elasticsearch:8.6.2
```

```
mkdir es-cluster
```

```
cd es-cluster
```

```
nano docker-compose.yml
```

```
services:
  es01:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.6.2
    container_name: es01
    environment:
      - node.name=es01
      - discovery.type=single-node
      - cluster.name=es-cluster
      - bootstrap.memory_lock=true
```

```
docker compose up -d
```

jso-analyzer, nori tokenizer 설치

```
docker exec -it es01
```

```
bin/elasticsearch-plugin install analysis-nori
```

```
bin/elasticsearch-plugin install https://github.com/netcrazy/elasticsearch-jso-analyzer/releases/download/v8.6.2/jso-analyzer-plugin-8.6.2-plugin.zip
```

docker restart es01

logstash 세팅

mkdir logstash

cd logstash

vi Dockerfile

```
FROM docker.elastic.co/logstash/logstash:8.17.2

# 기본 logstash.conf 삭제
RUN rm -f /usr/share/logstash/pipeline/logstash.conf

# JDBC 드라이버 복사
COPY jars/mysql-connector-j-8.0.33.jar /usr/share/logstash/lib/

# 파이프라인 설정 복사
COPY pipeline/*.conf /usr/share/logstash/pipeline

# logstash.yml 설정 복사
COPY logstash.yml /usr/share/logstash/config/logstash.yml
```

vi logstash.yml

```
xpack.monitoring.elasticsearch.hosts: [ "http://j12e202.p.ssafy.io:9200" ]
xpack.monitoring.enabled: true
```

mkdir pipeline

cd pipeline

vi feeds.conf

vi products.conf

vi users.conf

```
input {
  jdbc {
    jdbc_driver_library ⇒ "/usr/share/logstash/lib/mysql-connector-j-8.0.33.jar"
    jdbc_driver_class ⇒ "com.mysql.cj.jdbc.Driver"
    jdbc_connection_string ⇒ "jdbc:mysql://localhost:3306/keywi_feed"
```

jdbc_user ⇒ "mysql 유저"
jdbc_password ⇒ "mysql 비밀번호"
schedule ⇒ "* * * * *" # 매 분마다 실행
lowercase_column_names ⇒ false

```
statement ⇒ "  
SELECT  
  f.feed_id    AS feedId,  
  f.content    AS content,  
  f.created_at AS createdAt,  
  u.user_nickname AS userNickname,  
  u.profile_url AS userProfileImageUrl,  
  (  
    SELECT fi.image_url  
    FROM feed_images fi  
    WHERE fi.feed_id = f.feed_id AND fi.is_main_image = TRUE  
    ORDER BY fi.display_order ASC  
    LIMIT 1  
  ) AS thumbnailUrl,  
  
  (  
    SELECT JSON_ARRAYAGG(JSON_OBJECT('name', h.name))  
    FROM feed_hashtags fh  
    JOIN hashtags h ON fh.hashtag_id = h.hashtag_id  
    WHERE fh.feed_id = f.feed_id  
  ) AS hashtags,  
  
  (  
    SELECT JSON_ARRAYAGG(JSON_OBJECT(  
      'productId', fp.product_id,  
      'productName', fp.product_name,  
      'price', fp.price,  
      'categoryName', fp.category,  
      'thumbnailUrl', img.image_url  
    ))  
  FROM feed_products fp  
  LEFT JOIN feed_images img ON fp.image_id = img.image_id  
  WHERE fp.feed_id = f.feed_id
```

```

    ) AS taggedProducts

FROM feeds f
LEFT JOIN keywi.users u ON f.user_id = u.user_id
WHERE f.is_delete = FALSE
GROUP BY f.feed_id;

"
}
}

filter {
  json {
    source ⇒ "hashtags"
    target ⇒ "hashtags"
  }
  json {
    source ⇒ "taggedProducts"
    target ⇒ "taggedProducts"
  }
}

output {
  stdout { codec ⇒ rubydebug }

  elasticsearch {
    hosts ⇒ ["http://j12e202.p.ssafy.io:9200"]
    ssl ⇒ false
    user ⇒ "엘라스틱 유저"
    password ⇒ "엘라스틱 비밀번호"
    index ⇒ "feeds"
    document_id ⇒ "%{feedId}"
    action ⇒ "index"
    codec ⇒ json
  }
}

```



```

input {
  jdbc {
    jdbc_driver_library ⇒ "/usr/share/logstash/lib/mysql-connector-j-8.0.33.jar"
    jdbc_driver_class ⇒ "com.mysql.cj.jdbc.Driver"
    jdbc_connection_string ⇒ "jdbc:mysql://localhost:3306/keywi"

    jdbc_user ⇒ "mysql 유저"
    jdbc_password ⇒ "mysql 비밀번호"
    schedule ⇒ "* * * * *" # 매 분마다 실행
    lowercase_column_names ⇒ false

    statement ⇒ "
SELECT
  p.product_id AS productId,
  p.product_name AS productName,
  p.price,
  p.product_image AS imageUrl,
  p.manufacturer AS manufacturer,
  c.category_id AS categoryId,
  c.category_name AS categoryName,
  pc.category_id AS parentCategoryId,
  pc.category_name AS parentCategoryName
FROM products p
LEFT JOIN category c ON p.category_id = c.category_id
LEFT JOIN category pc ON c.parent_id = pc.category_id
"
  }
}

filter {
}

output {
  stdout { codec ⇒ rubydebug }

  elasticsearch {
    hosts ⇒ ["http://j12e202.p.ssafy.io:9200"]
  }
}

```

```

ssl ⇒ false
user ⇒ "엘라스틱 유저"
password ⇒ "엘라스틱 비밀번호"
index ⇒ "products"
document_id ⇒ "%{productId}"
action ⇒ "index"
codec ⇒ json
}
}

```

```

input {
  jdbc {
    jdbc_driver_library ⇒ "/usr/share/logstash/lib/mysql-connector-j-8.0.33.jar"
    jdbc_driver_class ⇒ "com.mysql.cj.jdbc.Driver"
    jdbc_connection_string ⇒ "jdbc:mysql://localhost:3306/keywi"

    jdbc_user ⇒ "mysql 유저"
    jdbc_password ⇒ "mysql 비밀번호"
    schedule ⇒ "* * * * *" # 매 분마다 실행
    lowercase_column_names ⇒ false

    statement ⇒ "
      SELECT
        u.user_id      AS userId,
        u.user_nickname AS nickname,
        u.status_message AS profileContent,
        u.brix         AS brix,
        u.profile_url AS profileImageUrl
      FROM users u
    "
  }
}

filter {
  mutate {
    convert ⇒ {
      "userId" ⇒ "integer"
    }
  }
}

```

```

    "brix" ⇒ "integer"
  }
}
}

output {
  stdout { codec ⇒ rubydebug }

  elasticsearch {
    hosts ⇒ ["http://j12e202.p.ssafy.io:9200"]
    ssl ⇒ false
    user ⇒ "엘라스틱 유저"
    password ⇒ "엘라스틱 비밀번호"
    index ⇒ "users"
    document_id ⇒ "%{userId}"
    action ⇒ "index"
    codec ⇒ json
  }
}

```

```
mkdir jars
```

```
cd jars
```

mysql-connector-j-8.0.33.jar 설치

```
docker compose up --build -d
```

 : conf 파일 수정될 때 실행해서 업데이트 해줘야 함

5. 백엔드 세팅

배포 순서(ES 따로) : config → eureka → gateway → auth → product → feed → mypage → financial → board → chat → search

bootstrap.yml 기본 세팅

```

spring:
  application:
    name: 서버 이름
  # Config Server 설정
  config:
    import: configserver:https://j12e202.p.ssafy.io/config

  # Spring Cloud Bus 설정 - 설정 재시작 허용
  cloud:
    bus:
      enabled: true
      refresh:
        enabled: true

  # Actuator 엔드포인트 설정
  management:
    endpoints:
      web:
        exposure:
          include: health,info,refresh,busrefresh #,gateway
    endpoint:
      health:
        show-details: never

  # 게이트웨이 서버
  gateway:
    enabled: true

```

이 파일을 사용하여 Config 서버에서 설정을 받아온다.

pom.xml 에 아래 설정이 되어 있어야 사용할 수 있다.

```

...
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>

```

```

</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-bus-amqp</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-bootstrap</artifactId>
</dependency>
</dependencies>

<dependencyManagement>
<dependencies>
  <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-dependencies</artifactId>
    <version>${spring-cloud.version}</version>
    <type>pom</type>
    <scope>import</scope>
  </dependency>
</dependencies>
</dependencyManagement>

```

5.1 Config 서버

```

server:
  port: 8888

spring:
  application:
    name: config-server

```

```

# RabbitMQ 설정 (Spring Cloud Bus)
rabbitmq:
  host: localhost
  username: {유저}
  password: {비밀번호}

# Spring Cloud Config 설정
cloud:
  config:
    server:
      git:
        uri: {깃 주소} # 로컬 저장소 경로, Git 서버 URL로 변경 가능
        default-label: master
        search-paths: "{application}" # 애플리케이션 이름과 정확히 일치하는 폴더
만 검색
        ignoreLocalSshSettings: true
        strictHostKeyChecking: false
        hostKey: {깃으로 생성한 키}
        hostKeyAlgorithm: ecdsa-sha2-nistp256
        privateKey: |
          {생성한 비밀 키}
# 암호화 설정 활성화
encrypt:
  enabled: false

# Spring Cloud Bus 설정
bus:
  enabled: true
  refresh:
    enabled: true
  env:
    enabled: true

# Actuator 엔드포인트 설정
management:
  endpoints:
    web:
      exposure:

```

```
    include: health,info,refresh,busrefresh
endpoint:
  health:
    show-details: never
```

```
# 암호화를 위한 대칭키 설정 (실제 환경에선 환경변수나 외부에서 관리)
encrypt:
  key: "{암호화 키}"
```

5.2 Eureka 서버

```
server:
  port: 8761
```

```
spring:
  application:
    name: eureka
```

```
# RabbitMQ 설정 (Spring Cloud Bus)
```

```
rabbitmq:
  host: localhost
  port: 5672
  username: {유저}
  password: {비밀번호}
```

```
# Eureka Server 설정
```

```
eureka:
  instance:
    hostname: localhost
    preferIpAddress: true
    instanceId: localhost:${server.port}
  client:
    registerWithEureka: false # 자신을 유레카 서버에 등록하지 않음
    fetchRegistry: false # 레지스트리 정보를 로컬에 캐싱하지 않음
    serviceUrl:
      defaultZone: http://${eureka.instance.hostname}:${server.port}/eureka/
  server:
```

```
waitTimeInMsWhenSyncEmpty: 5 # 서버가 요청을 처리하기 전 초기 대기 시간
enableSelfPreservation: false # 개발 환경에서는 false로 설정 (프로덕션에서는
true 권장)
```

```
# Actuator 설정
```

```
management:
```

```
  endpoints:
```

```
    web:
```

```
      exposure:
```

```
        include: health,info,refresh,busrefresh
```

```
  endpoint:
```

```
    health:
```

```
      show-details: never
```

```
      validate-group-membership: false # 단일 Eureka 서버 환경에서는 그룹 멤버
      십 검증 불필요
```

```
# Logging 설정 (Debugging 용)
```

```
logging:
```

```
  level:
```

```
    org.springframework.cloud.gateway: DEBUG
```

5.3 Gateway 서버

```
server:
```

```
  port: 8080
```

```
spring:
```

```
  application:
```

```
    name: gateway
```

```
# RabbitMQ 설정 (Spring Cloud Bus)
```

```
rabbitmq:
```

```
  host: localhost
```

```
  port: 5672
```

```
  username: {유저}
```

```
  password: {비밀번호}
```



```
servlet:
  multipart:
    max-file-size: 10MB
    max-request-size: 50MB
    enabled: true
# Cloud Gateway 설정
cloud:
  gateway:
    discovery:
      locator:
        enabled: true # 서비스 디스커버리를 통한 자동 라우팅 활성화
        lower-case-service-id: true # 서비스 ID를 소문자로 변환
    multipart:
      enabled: true
      max-file-size: 50MB
      max-request-size: 50MB

# 라우트 설정
routes:
  # 마이페이지
  - id: mypage
    uri: lb://mypage
    predicates:
      - Path=/api/ratings/**, /api/profile/**

#견적페이지
- id: board
  uri: lb://board
  predicates:
    - Path=/api/estimate-boards/**

# 인증(Auth) 서비스
- id: auth
  uri: lb://auth
  predicates:
    - Path=/api/auth/**

# 채팅(Chat) 서비스
```

```
- id: chat
uri: lb://chat
predicates:
  - Path=/api/chat/**, /ws-endpoint/**, /app/**
```

피드(Feeds) 서비스 웹소켓이라 nginx에서 바로 연결할지 게이트웨이 거칠지
고민

```
- id: feed
uri: lb://feed
predicates:
  - Path=/api/feed/**
```

상품(Products) 서비스

```
- id: product
uri: lb://product
predicates:
  - Path=/api/product/**
```

결제(Payments) 서비스

```
- id: financial
uri: lb://financial
predicates:
  - Path=/api/financial/**, /api/payment/**, /api/bank/**
```

검색(Search) 서비스

```
- id: search
uri: lb://search
predicates:
  - Path=/api/search/**, /api/autocomplete/**
```

jwt:

```
secret: { jwt-secret키 }
```

Eureka 클라이언트 설정

eureka:

client:

service-url:

defaultZone: http://localhost:8761/eureka/

```
instance:
  prefer-ip-address: true
  instance-id: ${spring.application.name}:${server.port}
  hostname: localhost
```

Actuator 설정

```
management:
  endpoints:
    web:
      exposure:
        include: health,info,refresh,busrefresh,gateway
  endpoint:
    health:
      show-details: never
  gateway:
    enabled: true
```

5.4 Auth 서버

```
server:
  port: 8100 # 인증 서비스가 실행될 포트 번호

spring:
  application:
    name: auth # 이 애플리케이션의 이름 (대문자로 설정, Eureka에 이 이름으로 등록됨)\

# RabbitMQ 설정 (Spring Cloud Bus)
rabbitmq:
  host: localhost
  port: 5672
  username: {유저}
  password: {비밀번호}

datasource: # MySQL 데이터베이스 연결 설정
  url: jdbc:mysql://localhost:3306/keywi?createDatabaseIfNotExist=true&serverTimezone=Asia/Seoul&characterEncoding=UTF-8 # DB URL (없으면 생
```

성)

username: {mysql 유저}
password: {비밀번호}
driver-class-name: com.mysql.cj.jdbc.Driver # MySQL JDBC 드라이버

jpa:

hibernate:
ddl-auto: update # 데이터베이스 스키마 자동 업데이트 (개발 환경에 적합)
show-sql: true # SQL 쿼리를 콘솔에 출력

redis: # Redis 설정 (세션 또는 토큰 저장소로 사용될 수 있음)

host: localhost # Redis 서버 호스트
port: 6379 # Redis 서버 포트

oauth2: # OAuth2 카카오 로그인 설정

kakao:

client_id: {카카오 클라이언트 아이디}
redirect_uri: {카카오에 등록된 uri}/callback/kakao

eureka:

client:

service-url:

defaultZone: http://localhost:8761/eureka/ # Eureka 서버의 위치 (서비스 등록 위치)

instance:

instance-id: \${spring.application.name}:\${random.uuid} # 인스턴스의 고유 ID 생성 (애플리케이션 이름 + 랜덤 UUID)

JWT 토큰 설정

jwt:

secret: {jwt 토큰 키} # JWT 서명에 사용되는 비밀 키
access-token-validity: 3600000 # 액세스 토큰 유효 기간 (1시간, 밀리초 단위)
expiration: 86400000 # JWT 토큰 만료 시간 (24시간, 밀리초 단위)
refresh-token-validity: 604800000 # 리프레시 토큰 유효 기간 (7일, 밀리초 단위)

logging:

level:

com.ssafy.auth: DEBUG # 인증 서비스 관련 로그를 상세히 출력

```
app:
  default-profile-image: https://key-wi.s3.ap-northeast-2.amazonaws.com/
  profiles/default_profile.png

# AWS S3 설정
cloud:
  aws:
    credentials:
      access-key: {s3 액세스 키}
      secret-key: {s3 시크릿 키}
    s3:
      bucket: {버킷 이름}
    region:
      static: ap-northeast-2
    stack:
      auto: false
```

5.5 Product 서버

```
server:
  port: 8500 # 인증 서비스가 실행될 포트 번호

spring:
  application:
    name: product # 이 애플리케이션의 이름 (Eureka에 이 이름으로 등록됨)

# RabbitMQ 설정 (Spring Cloud Bus)
rabbitmq:
  host: localhost
  port: 5672
  username: {유저}
  password: {비밀번호}

datasource: # MySQL 데이터베이스 연결 설정
  url: jdbc:mysql://localhost:3306/keywi?createDatabaseIfNotExist=true&serverTimezone=Asia/Seoul&characterEncoding=UTF-8 # DB URL (없으면 생성)
```

```
username: {mysql 유저}
password: {비밀번호}
driver-class-name: com.mysql.cj.jdbc.Driver # MySQL JDBC 드라이버
jpa:
  hibernate:
    ddl-auto: update # 데이터베이스 스키마 자동 업데이트 (개발 환경에 적합)
    show-sql: true # SQL 쿼리를 콘솔에 출력
redis: # Redis 설정 (세션 또는 토큰 저장소로 사용될 수 있음)
  host: localhost # Redis 서버 호스트
  port: 6379 # Redis 서버 포트

eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/ # Eureka 서버의 위치 (서비스
      등록 위치)
  instance:
    instance-id: ${spring.application.name}:${random.uuid} # 인스턴스의 고유
    ID 생성 (애플리케이션 이름 + 랜덤 UUID)

mybatis:
  mapper-locations: classpath:mappers/*.xml
  configuration:
    map-underscore-to-camel-case: true

logging:
  level:
    com.ssafy.product: DEBUG # 인증 서비스 관련 로그를 상세히 출력
```

5.6 Feed 서버

```
server:
  port: 8400

spring:
  application:
    name: feed
```

```

# 데이터베이스 설정
datasource:
  url: jdbc:mysql://localhost:3306/keywi_feed?useSSL=false&serverTimez
one=Asia/Seoul&characterEncoding=UTF-8
  username: {mysql 사용자}
  password: {비밀번호}
  driver-class-name: com.mysql.cj.jdbc.Driver
# Cloud Bus 설정
cloud:
  bus:
    enabled: true
    refresh:
      enabled: true

# RabbitMQ 설정 (Config Server Bus)
rabbitmq:
  host: localhost
  port: 5672
  username: {사용자}
  password: {비밀번호}

servlet:
  multipart:
    max-file-size: 10MB
    max-request-size: 50MB
    enabled: true
# Kafka 설정
kafka:
  bootstrap-servers: localhost:9092
  consumer:
    group-id: feed-service
    auto-offset-reset: earliest
    key-deserializer: org.apache.kafka.common.serialization.StringDeseriali
zer
    value-deserializer: org.springframework.kafka.support.serializer.JsonD
eserializer
  properties:
    spring.json.trusted.packages: "*"

```

```
producer:
  key-serializer: org.apache.kafka.common.serialization.StringSerializer
  value-serializer: org.springframework.kafka.support.serializer.JsonSerializer
listener:
  missing-topics-fatal: false
# Redis 설정
redis:
  host: localhost
  port: 6380

# MyBatis 설정
mybatis:
  mapper-locations: classpath:mappers/**/*.xml
  type-aliases-package: com.ssafy.feed.model
  configuration:
    map-underscore-to-camel-case: true
    call-setters-on-nulls: true
    jdbc-type-for-null: "NULL"
    default-executor-type: REUSE

# Eureka 클라이언트 설정
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
  instance:
    prefer-ip-address: true
    instance-id: ${spring.application.name}:${server.port}

# Actuator 설정
management:
  endpoints:
    web:
      exposure:
        include: "*"
  endpoint:
    health:
```



```

show-details: never

cloud:
  aws:
    credentials:
      access-key: {AWS 액세스 키} # 환경 변수에서 읽거나 직접 입력
      secret-key: {AWS 시크릿 키} # 환경 변수에서 읽거나 직접 입력
    region:
      static: ap-northeast-2 # 서울 리전
    s3:
      bucket: {버킷 이름} # 환경 변수에서 읽거나 직접 입력
    stack:
      auto: false
  resilience4j:
    circuitbreaker:
      instances:
        productService:
          registerHealthIndicator: true
          slidingWindowSize: 10
          minimumNumberOfCalls: 5
          permittedNumberOfCallsInHalfOpenState: 3
          automaticTransitionFromOpenToHalfOpenEnabled: true
          waitDurationInOpenState: 5s
          failureRateThreshold: 50
          eventConsumerBufferSize: 10
    timelimiter:
      instances:
        productService:
          timeoutDuration: 3s
# 피드 서비스 설정
feed:
  recommendation:
    schedule:
      personal: "0 */30 * * * *" # 30분마다 실행 (cron 표현식)
      popular: "0 0 */1 * * *" # 1시간마다 실행
    cache:
      ttl: 86400 # 24시간 (초 단위)
  storage:

```

```
upload-dir: ./uploads/feed-images # 이미지 업로드 디렉토리
feign:
  circuitbreaker:
    enabled: true
```

```
server:
  port: 8400

spring:
  application:
    name: feed
  # 데이터베이스 설정
  datasource:
    url: jdbc:mysql://localhost:3306/keywi_feed?useSSL=false&serverTimez
one=Asia/Seoul&characterEncoding=UTF-8
    username: {mysql 유저}
    password: {비밀번호}
    driver-class-name: com.mysql.cj.jdbc.Driver
  # Cloud Bus 설정
  cloud:
    bus:
      enabled: true
      refresh:
        enabled: true

  # RabbitMQ 설정 (Config Server Bus)
  rabbitmq:
    host: localhost
    port: 5672
    username: {유저}
    password: {비밀번호}

servlet:
  multipart:
    max-file-size: 10MB
    max-request-size: 50MB
    enabled: true
  # Kafka 설정
```

```
kafka:
  bootstrap-servers: localhost:9092
  consumer:
    group-id: feed-service
    auto-offset-reset: earliest
    key-deserializer: org.apache.kafka.common.serialization.StringDeseriali
zer
    value-deserializer: org.springframework.kafka.support.serializer.JsonD
eserializer
  properties:
    spring.json.trusted.packages: "*"
  producer:
    key-serializer: org.apache.kafka.common.serialization.StringSerializer
    value-serializer: org.springframework.kafka.support.serializer.JsonSeri
alizer
  listener:
    missing-topics-fatal: false
# Redis 설정
redis:
  host: localhost
  port: 6380

# MyBatis 설정
mybatis:
  mapper-locations: classpath:mappers/**/*.xml
  type-aliases-package: com.ssafy.feed.model
  configuration:
    map-underscore-to-camel-case: true
    call-setters-on-nulls: true
    jdbc-type-for-null: "NULL"
    default-executor-type: REUSE

# Eureka 클라이언트 설정
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
  instance:
```

```

prefer-ip-address: true
instance-id: ${spring.application.name}:${server.port}

# Actuator 설정
management:
  endpoints:
    web:
      exposure:
        include: "*"
  endpoint:
    health:
      show-details: never

cloud:
  aws:
    credentials:
      access-key: {AWS 액세스 키} # 환경 변수에서 읽거나 직접 입력
      secret-key: {AWS 시크릿 키} # 환경 변수에서 읽거나 직접 입력
    region:
      static: ap-northeast-2 # 서울 리전
    s3:
      bucket: {버킷이름} # 환경 변수에서 읽거나 직접 입력
    stack:
      auto: false
  resilience4j:
    circuitbreaker:
      instances:
        productService:
          registerHealthIndicator: true
          slidingWindowSize: 10
          minimumNumberOfCalls: 5
          permittedNumberOfCallsInHalfOpenState: 3
          automaticTransitionFromOpenToHalfOpenEnabled: true
          waitDurationInOpenState: 5s
          failureRateThreshold: 50
          eventConsumerBufferSize: 10
    timelimiter:
      instances:

```

```

    productService:
      timeoutDuration: 3s
# 피드 서비스 설정
feed:
  recommendation:
    schedule:
      personal: "0 */30 * * * *" # 30분마다 실행 (cron 표현식)
      popular: "0 0 */1 * * *" # 1시간마다 실행
    cache:
      ttl: 86400 # 24시간 (초 단위)
  storage:
    upload-dir: ./uploads/feed-images # 이미지 업로드 디렉토리
  feign:
    circuitbreaker:
      enabled: true

```

5.7 Mypage 서버

```

server:
  port: 8600

spring:
  application:
    name: mypage

  datasource:
    url: jdbc:mysql://localhost:3306/keywi?serverTimezone=Asia/Seoul&characterEncoding=UTF-8
    username: {mysql 유저}
    password: {비밀번호}
    driver-class-name: com.mysql.cj.jdbc.Driver

  mvc:
    pathmatch:
      matching-strategy: ant_path_matcher

  rabbitmq:

```

```
host: localhost
port: 5672
username: {유저}
password: {비밀번호}
```

```
cloud:
  bus:
    enabled: true
    refresh:
      enabled: true
```

```
mybatis:
  mapper-locations: classpath:mapper/**/*.xml
  type-aliases-package: com.ssafy.mypage.profile.dto
  configuration:
    map-underscore-to-camel-case: true
```

```
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/ # Eureka 서버의 위치 (서비스
등록 위치)
  instance:
    instance-id: ${spring.application.name}:${random.uuid} # 인스턴스의 고유
ID 생성 (애플리케이션 이름 + 랜덤 UUID)
```

```
logging:
  level:
    root: info
    com.ssafy: debug
```

5.8 Financial 서버

```
server:
  port: 8030

spring:
```

```
application:
  name: financial

datasource:
  url: jdbc:mysql://localhost:3306/financial?serverTimezone=Asia/Seoul&characterEncoding=UTF-8
  username: {mysql 유저}
  password: {비밀번호}
  driver-class-name: com.mysql.cj.jdbc.Driver
  hikari:
    maximum-pool-size: 10
    minimum-idle: 5
    idle-timeout: 300000
    connection-timeout: 20000

rabbitmq:
  host: localhost
  port: 5672
  username: {유저}
  password: {비밀번호}

cloud:
  bus:
    enabled: true
  refresh:
    enabled: true

financial:
  api:
    url: https://finopenapi.ssafy.io/ssafy/api/v1
    api-key: 싸피 금융 api키

escrow:
  account:
    number: 에스크로 계좌
    bank-code: 에스크로 계좌 은행코드
# 유레카 클라이언트 설정
eureka:
```

```
client:
  service-url:
    defaultZone: http://localhost:8761/eureka/
instance:
  prefer-ip-address: true
  instance-id: ${spring.application.name}:${spring.application.instance_id:${random.value}}
```

5.9 Board 서버

```
server:
  port: 8300
  servlet:
    context-path: /

spring:
  application:
    name: board

  # RabbitMQ 설정 (Spring Cloud Bus)
  rabbitmq:
    host: localhost
    port: 5672
    username: {유저}
    password: {비밀번호}

  # 데이터베이스 설정
  datasource:
    url: jdbc:mysql://localhost:3306/keywi?createDatabaseIfNotExist=true&useSSL=false&serverTimezone=Asia/Seoul&characterEncoding=UTF-8
    username: {mysql 유저}
    password: {비밀번호}
    driver-class-name: com.mysql.cj.jdbc.Driver

  sql:
    init:
      mode: always
```



```
platform: mysql

# 파일 업로드 설정
servlet:
  multipart:
    max-file-size: 10MB
    max-request-size: 50MB

# MyBatis 설정
mybatis:
  mapper-locations: classpath:mapper/**/*.xml
  type-aliases-package: com.ssafy.board.entity
  configuration:
    map-underscore-to-camel-case: true

# 유레카 클라이언트 설정
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
  instance:
    prefer-ip-address: true
    instance-id: ${spring.application.name}:${spring.application.instance_id:${random.value}}

# 로깅 설정
logging:
  level:
    com.ssafy.board: DEBUG
    org.mybatis: DEBUG
    org.springframework.web: INFO

# AWS S3 설정
cloud:
  aws:
    credentials:
      access-key: {s3 액세스 키}
      secret-key: {s3 시크릿 키}
```

```
s3:
  bucket: key-wi
  endpoint: https://key-wi.s3.ap-northeast-2.amazonaws.com
  region:
    static: ap-northeast-2
  stack:
    auto: false
```

5.10 Chat 서버

```
server:
  port: 8082 # 채팅 서비스 포트

spring:
  application:
    name: chat

# 멀티파트 파일 업로드 설정 추가
servlet:
  multipart:
    enabled: true
    max-file-size: 10MB
    max-request-size: 50MB
    file-size-threshold: 2MB

# Spring Cloud Config 비활성화
cloud:
  bus:
    enabled: true
    refresh:
      enabled: true

# RabbitMQ 설정 (Config Server Bus)
rabbitmq:
  host: localhost
  port: 5672
  username: {유저}
```

```

password: {비밀번호}

# MySQL 설정
datasource:
  url: jdbc:mysql://localhost:3306/keywi?createDatabaseIfNotExist=true&serverTimezone=Asia/Seoul&characterEncoding=UTF-8
  username: {mysql 유저}
  password: {비밀번호}
  driver-class-name: com.mysql.cj.jdbc.Driver

jpa:
  hibernate:
    ddl-auto: update
  properties:
    hibernate:
      dialect: org.hibernate.dialect.MySQL8Dialect
# MongoDB 설정
data:
  mongodb:
    host: localhost
    port: 27017
    database: keyboard-chat

# Kafka 설정
kafka:
  bootstrap-servers: localhost:9092
  consumer:
    group-id: chat-group
    auto-offset-reset: latest
    key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
    value-deserializer: org.springframework.kafka.support.serializer.JsonDeserializer
  properties:
    spring.json.trusted.packages: "*"
  producer:
    key-serializer: org.apache.kafka.common.serialization.StringSerializer
    value-serializer: org.springframework.kafka.support.serializer.JsonSerializer

```

```

alizer
  listener:
    missing-topics-fatal: false
# Eureka 설정
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/
    # Eureka 서버가 없는 경우 아래 설정 추가 가능
    # register-with-eureka: false
    # fetch-registry: false
  instance:
    instance-id: ${spring.application.name}:${server.port}
    prefer-ip-address: true

# FCM 설정
fcm:
  enabled: false
  service-account-file: classpath:firebase-service-account.json

logging:
  level:
    org.springframework.web.socket: TRACE
    org.springframework.messaging: TRACE
    com.ssafy.chat: TRACE

# AWS S3 설정
cloud:
  aws:
    credentials:
      access-key: {s3 액세스 키}
      secret-key: {s3 시크릿 키}
    s3:
      bucket: key-wi
    region:
      static: ap-northeast-2
  stack:
    auto: false

```

5.11 Search 서버

```
spring:
  application:
    name: search

# main:
# allow-bean-definition-overriding: true

# Elasticsearch 연결 정보
config:
  elastic:
    host: j12e202.p.ssafy.io
    port: 9200

# RabbitMQ 설정 (Spring Cloud Bus)
rabbitmq:
  host: localhost
  port: 5672
  username: {유저}
  password: {비밀번호}

data:
  redis:
    host: localhost
    port: 6379

batch:
  jdbc:
    initialize-schema: always # 스키마 자동 생성 (초기 개발 시만)
  job:
    enabled: false # 자동 실행 방지 (스케줄러로 수동 실행)

datasource:
  url: jdbc:mysql://localhost:3306/keywi?serverTimezone=Asia/Seoul&characterEncoding=UTF-8
  username: {mysql 유저}
  password: {비밀번호}
```

```
driver-class-name: com.mysql.cj.jdbc.Driver
hikari:
  maximum-pool-size: 10
  minimum-idle: 5
  idle-timeout: 300000
  connection-timeout: 20000

eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/ # Eureka 서버의 위치 (서비스
등록 위치)
  instance:
    instance-id: ${spring.application.name}:${random.uuid} # 인스턴스의 고유
ID 생성 (애플리케이션 이름 + 랜덤 UUID)

mybatis:
  mapper-locations: classpath:mapper/**/*.xml
  type-aliases-package: com.ssafy.search.dto
  configuration:
    map-underscore-to-camel-case: true
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl

server:
  port: 8200
  servlet:
    context-path: /
  tomcat:
    max-threads: 200
    min-spare-threads: 30

logging:
  level:
    com.ssafy: DEBUG
    org.springframework: INFO
    org.elasticsearch: INFO
    org.springframework.batch: DEBUG
    ssafy.com.ssafy.search: DEBUG
```

6. 외부 서비스

6.1 소셜 로그인 - Kakao

6.1.1 카카오 로그인 활성화

카카오 로그인 **ON**

[동의 화면 미리보기](#)

활성화 설정

상태

ON

카카오 로그인 API를 활용하면 사용자들이 번거로운 회원 가입 절차 대신, 카카오톡으로 서비스를 시작할 수 있습니다.

상태가 OFF일 때도 카카오 로그인 설정 항목을 변경하고 서버에 저장할 수 있습니다.

상태가 ON일 때만 실제 서비스에서 카카오 로그인 화면이 연결됩니다.

6.1.2 리다이렉트 UR설정

Redirect URI

삭제

수정

Redirect URI	
	https://j12e202.p.ssafy.io/api/auth/callback/kakao
	http://j12e202.p.ssafy.io/api/auth/callback/kakao
	http://localhost:8080/api/auth/callback/kakao
	http://localhost:5173/callback/kakao
	https://keywi.polocestelle.site/callback/kakao

- 카카오 로그인에서 사용할 OAuth Redirect URI를 설정합니다. (최대 10개)
- REST API로 개발하는 경우 필수로 설정해야 합니다.

6.1.3 요청 흐름

1. 프론트엔드 로그인 요청 주소

[카카오]

https://kauth.kakao.com/oauth/authorize?client_id={카카오 클라이언트 아이디}&redirect_uri={리다이렉트 uri}

2. 로그인 요청 주소로 인가코드를 요청

3. 프론트엔드에서 리다이렉트 uri 로 넘어온 code = {authCode} 를 통해서 백엔드 요청을 보냄
4. 백엔드에서 넘어온 authCode 를 이용하여 카카오에 정보를 요청하여 설정한 정보들을 받아옴
5. 정보들을 통해 db에 해당 유저가 없으면 유저를 생성해주고, 회원가입을 진행함
6. JWT accessToken을 가지고 redis에 저장한 정보와 일치하는지 확인하여 우리 서비스의 로그인을 진행함
7. 백엔드에서 넘겨준 JWT accessToken으로 우리 앱에서 로그인 및 회원가입 진행