포팅 메뉴얼

환경변수(application-secret.yml)

```
#resources/application-secret.yml
spring:
  datasource:
    url: jdbc:mysql://i12d101.p.ssafy.io:3306/mafia
    username: 유저이름
    password: 비밀번호
    driver-class-name: com.mysql.cj.jdbc.Driver
    data:
    redis:
      password: 비밀번호

hantu-openapi:
    domain: https://openapi.koreainvestment.com:9443 #실전도메인
    appkey: api_appkey
    appsecret: appsecret

hantu-openapi-second:
  domain: https://openapi.koreainvestment.com:9443
  appkey:
  appsecret: appsecret

#aws s3 bucket
cloud:
```

```
  aws:
    s3:
      bucket: bdl-image-bucket
    credentials:
      access-key: api_appkey
      secret-key: appsecret+nUAGb6mbmg8dyU9paCG22Z
    region:
      static: ap-southeast-2
      auto: false
    stack:
      auto: false


openai:
  api:
    key: api_appkey
```

빌드 배포 메뉴얼

1. EC2 인스턴트 준비

1. AWS Ubuntu EC2 인스턴트를 생성
2. 필요한 포트 열기
    - Jenkins: `8080`
    - SSH: `22`
    - 애플리케이션 포트(예: `80`, `443`, `3000` 등).

2. pem키 준비

1. `.pem` 파일을 로컬 컴퓨터에 저장

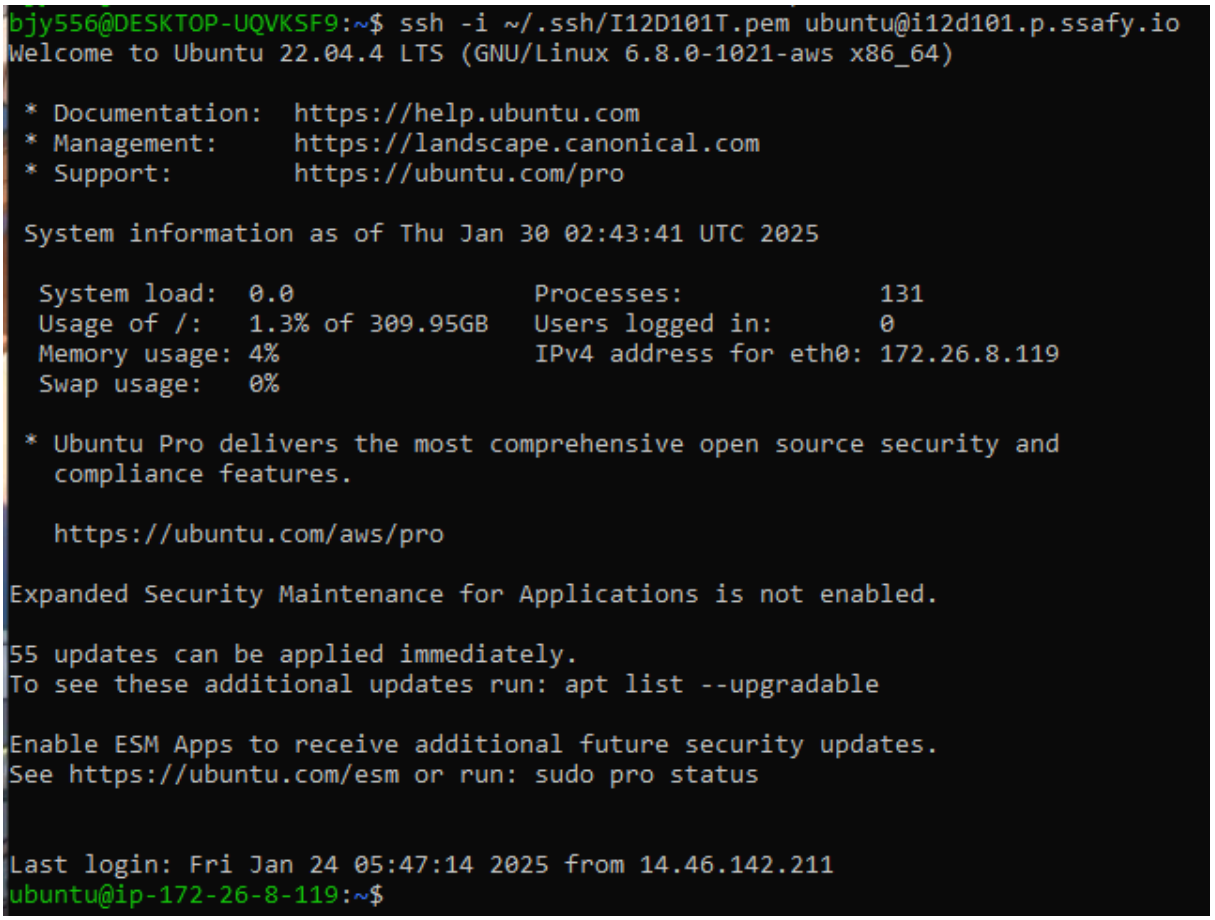
```
#PowerShell
wsl --install -d Ubuntu
```

2. ubuntu 계정에 대한 비밀번호 설정

```
sudo passwd
su root
```

3. 서버 접속

```
#폴더 생성
mkdir .ssh
# 자신 경로에 pem키 있는지 확인
cd /mnt/c/Users/Bae/Desktop/ssafy/pem
#해당 경로로 복사
cp /mnt/c/Users/Bae/Desktop/ssafy/pem/J12D202T.pem ~/.ssh/
#권한 설정(읽기)
chmod 400 ~/.ssh/J12D202T.pem
#서버 접속
ssh -i ~/.ssh/J12D202T.pem ubuntu@j12d202.p.ssafy.io
```



3. 우분투 방화벽 설정

```
# 필수 포트 허용
sudo ufw allow 22    # SSH
sudo ufw allow 80    # HTTP
sudo ufw allow 443   # HTTPS
sudo ufw allow 8080  # Jenkins
sudo ufw allow 8081  # Back Server
sudo ufw allow 3000  # grafana
```

4. 도커설치

4.1. Docker 레포지토리 설정

```
# 시스템의 패키지 목록을 최신화
sudo apt-get update

# SSL 인증서와 curl 도구 설치 (보안 통신과 파일 다운로드에 필요)
sudo apt-get install ca-certificates curl

# Docker의 GPG 키를 저장할 디렉토리 생성 (권한: 0755)
sudo install -m 0755 -d /etc/apt/keyrings

# Docker의 공식 GPG 키를 다운로드 (패키지 인증에 사용)
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/ke

# 다운로드한 GPG 키를 모든 사용자가 읽을 수 있도록 권한 설정
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

4.2. 레포지토리 추가

```
# Docker 공식 레포지토리를 시스템의 소프트웨어 소스에 추가
# - arch=$(dpkg --print-architecture): 시스템 아키텍처 확인 (예: amd64)
# - VERSION_CODENAME: Ubuntu 버전 코드네임 (예: focal)
echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/do
```

4.3. Docker 패키지 설치

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

4.4. Docker 설치 확인 및 권한 설정

```
# 현재 사용자를 docker 그룹에 추가
sudo usermode -aG docker $USER
#변경사항 적용
newgrp docker
#권한 확인
groups
`ubuntu adm dialout cdrom floppy sudo audio dip video plugdev netdev lxd dc
```

5. Jenkins 설치

5.1. 호스트 특정 디렉토리에 마운트

```
cd /home/ubuntu && mkdir jenkins-data
```

5.2. Jenkins Docker 컨테이너에 설치

```
docker run -d \
  -v /home/ubuntu/jenkins-data:/var/jenkins_home \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /home/ubuntu/docker/proxy:/proxy \
  -p 8080:8080 \
  -e JENKINS_OPTS="--prefix=/jenkins" \
  --group-add $(getent group docker | cut -d: -f3) \
  -e TZ=Asia/Seoul \
  --restart=on-failure \
  --name jenkins \
  jenkins/jenkins:lts-jdk17

  # -v 호스트의 /home/ubuntu/jenkins-data 디렉토리를 컨테이너의 /var/jenkins_hc
  # -v 호스트의 Docker 소켓을 컨테이너에 마운트
  # -v 호스트의 해당 폴더로 마운트
```

```
  # -e Jenkins의 URL 접두사를 '/jenkins'로 설정
  # -e Docker 명령어를 젠킨스 내에서 실행할 권한을 부여
  # -- 실패했을 경우 재시작
  # -- jenkins로 이름 지정
  # -- JDK17버전을 명시적으로 지정
```

초기 비밀번호 확인 `docker logs jenkins`

5.3. Jenkins 환경설정

```
cd /home/ubuntu/jenkins-data

mkdir update-center-rootCAs
#Jenkins가 업데이트 센터에 접속할 때 사용할 SSL 인증서를 제공
wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-center/rootCA/update-
#Jenkins가 기본 업데이트 센터 대신 Tencent 미러를 사용하도록 설정
sudo sed -i 's#https://updates.jenkins.io/update-center.json#https://raw.githu
#그 후 재시작
sudo docker restart jenkins
```

5.4. config 보안 설정 확인

```
vi config.xml
#true가 되어 있어야함
<useSecurity>true</useSecurity>
<securityRealm class="hudson.security.HudsonPrivateSecurityRealm">
<disableSignup>true</disableSignup>
```

5.5. Jenkins 초기 설정

1. http://j12d202.p.ssafy.io:8080 에 접속

2. Install suggeted plugins : 초기 플러그인 모두 설치
3. Getting Started - 계정 생성
4. 접속 주소

5.6. Jenkins내 docker명령어 실행

- DooD 방식

1. **Jenkins 안에 Docker를 설치하기 위해서 Jenkins 컨테이너에 접속**

```
docker exec -it -u root jenkins bash
```

2. **Jenkins 안에 Docker를 설치**

```
# 필요한 패키지 설치
apt-get update
apt-get install -y \
    ca-certificates \
    curl \
    gnupg \
    lsb-release

# Docker의 공식 GPG 키 추가
mkdir -p /etc/apt/keyrings
```

```
curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o

# Docker repository 설정
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker
  $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null

# 패키지 목록 업데이트
apt-get update

# Docker CLI만 설치
apt-get install -y docker-ce-cli
```

5.7. Credentials 저장

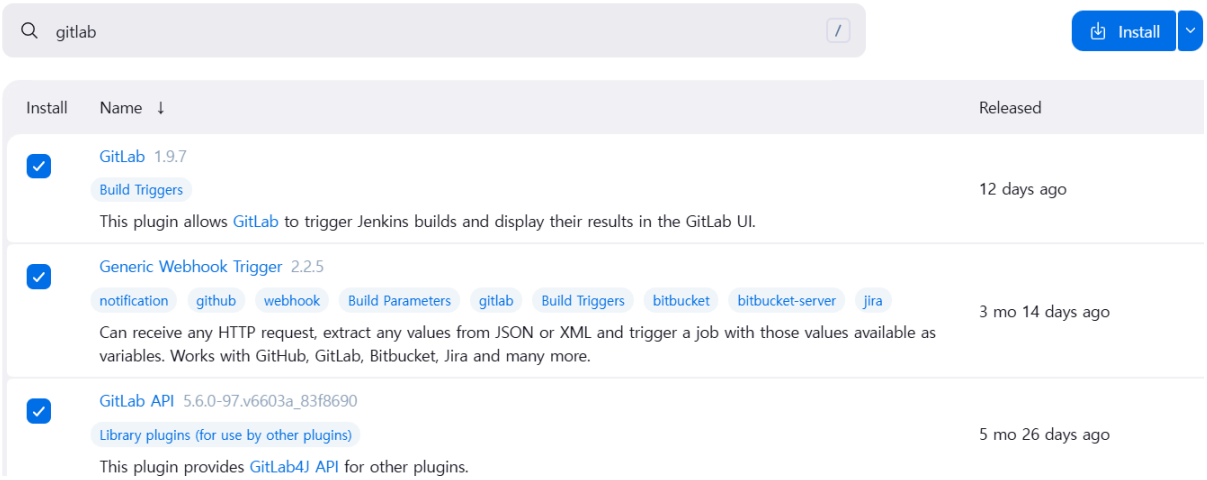
1. .env 작성

```
MYSQL_ROOT_PASSWORD=시크릿키
MYSQL_DATABASE=BLD
MYSQL_USER=BLD
MYSQL_PASSWORD=시크릿키
```

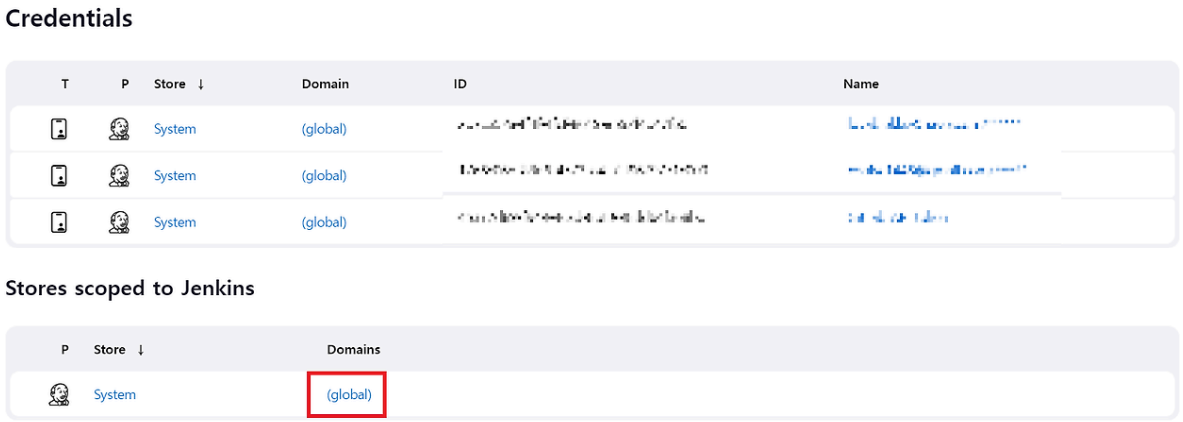
6. Jenkins, Gitlab 연동하기

6.1. Jenkins plugin 설치

Jenkins관리 → Plugins 클릭



6.2. Credential 등록

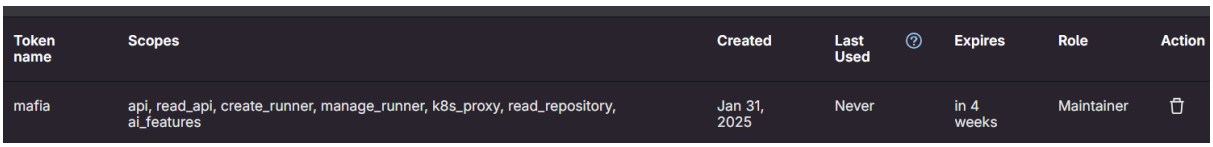


6.3. Gitlab 연결하기

Gitlab 계정 - Settings - Access Tokens 발급

→ 프로젝트 토큰 발급이 아니라 개인 계정 토큰 발급 필수 !!

Access Token



Jenkins 관리 - Credentials - global - add credentials

user: bjy556@naver.com
password : Access Token

api token으로 한번 더 credential 만들기

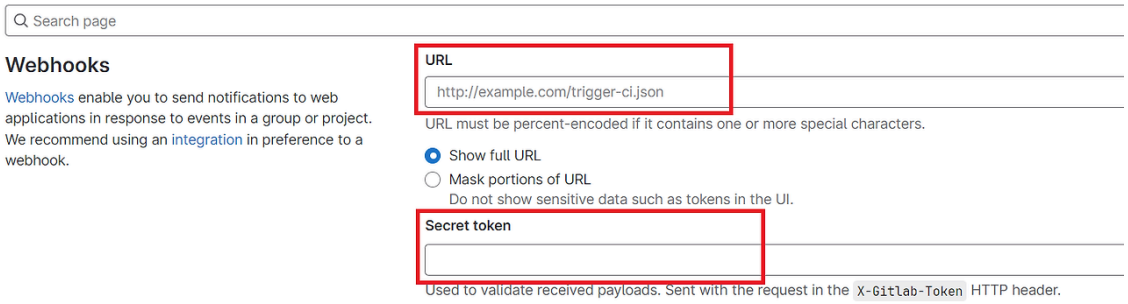
6.4. Pipeline 생성 및 Webhook 연결

jenkins : 새로운 item → pipeline

- Generate 버튼을 클릭 후
  - jenkins secret token 생성

GitLab : 프로젝트 → setting → webhook

- URL과 jenkins Secret token 입력



7. MySQL, Redis 컨테이너 설치

7.1 docker-compose.yml 작성

```
#home/ubuntu/docker/db/docker-compose.yml
services:
  mysql:
    image: mysql:8
    container_name: mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      MYSQL_DATABASE: ${MYSQL_DATABASE}
      MYSQL_USER: ${MYSQL_USER}
      MYSQL_PASSWORD: ${MYSQL_PASSWORD}
    ports:
      - "3306:3306"
    volumes:
      - ./mysql/init.sql:/docker-entrypoint-initdb.d/init.sql
      - mysql_data:/var/lib/mysql
    command:
      - --character-set-server=utf8mb4
      - --collation-server=utf8mb4_unicode_ci
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      interval: 10s
      timeout: 5s
```

```
      retries: 5
    networks:
      - app-network

  redis:
    image: redis:7
    container_name: redis
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data
    command: redis-server --requirepass 'Password' --appendonly yes
    networks:
      - app-network

volumes:
  mysql_data:
  redis_data:

networks:
  app-network:
    external: true
```

7.2. .env 설정

```
#docker/db/.env
MYSQL_ROOT_PASSWORD=시크릿키
MYSQL_DATABASE=BLD
MYSQL_USER=BLD
MYSQL_PASSWORD=시크릿키
```

7.3. docker 명령어 실행

```
#해당 폴더 위치로 이동
cd home/ubuntu/docker/db/docker-compose.yml

docker compose up -d
```

7.4 DB init 설정하기

```
#MYSQL 컨테이너 접속
docker exec -it mysql bash
#MYSQL에 root로 로그인
mysql -u root -p
#password 입력
ssafyD202!
#권한 부여 명령어 실행
USE BLD;
GRANT ALL PRIVILEGES ON BLD.* TO 'BLD'@'%;
FLSUH PRIVILEGES;

#확인하기
SHOW GRANTS FOR 'BLD'@'%';
#나가기
exit;
```

8. NGINX 컨테이너 설치

8.1 docker-compose.yml 작성

```
#home/ubuntu/docker/proxy/docker-compose.yml
services:
  nginx:
    image: nginx:latest
    container_name: nginx
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot
    command: "/bin/sh -c 'while :; do sleep 6h & wait $${!}; nginx -s reload; do
```

```
    networks:
      - app-network
    restart: always


  certbot:
    image: certbot/certbot
    volumes:
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot
    entrypoint: "/bin/sh -c 'trap exit TERM; while :; do certbot renew; sleep 12h
    depends_on:
      - nginx
    networks:
      - app-network

networks:
  app-network:
    external: true
```

8.2 CertBot https 인증서 발급

8.2.1 nginx.conf 작성

```
 upstream backend {
    server blue:8081;
    server green:8082 backup;
}

server {
    listen 80;
    listen [::]:80;
    server_name i12d101.p.ssafy.io;

    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }
```

```
    location / {
        return 301 https://$server_name$request_uri;
    }
}
```

8.2.2 폴더 생성 및 권한 설정

```
mkdir -p data/certbot/conf
mkdir -p data/certbot/www
sudo chown -R ubuntu:ubuntu data/certbot
sudo chmod -R 755 data/certbot

//인증서 발급 받기
docker compose exec certbot certbot certonly --webroot -w /var/www/certbo

Saving debug log to /var/log/letsencrypt/letsencrypt.log
```

8.2.4 인증서 발급이 성공되면 SSL pem 파일 작성

```
openssl dhparam -out data/certbot/conf/ssl-dhparams.pem 2048
```

8.3 Nginx 작성

```
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log warn;
pid /var/run/nginx.pid;

events {
    worker_connections 1024;
}

http {
    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    large_client_header_buffers 4 256k;
```

```
upstream backend {
    server spring:8081;
}

upstream frontend {
    server react:80;
}

server {
    listen 80;
    listen [::]:80;
    server_name j12d202.p.ssafy.io;

    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
    }

    location / {
        return 301 https://$server_name$request_uri;
    }
}

server {
    listen 443 ssl;
    server_name j12d202.p.ssafy.io;
    server_tokens off;

    ssl_certificate /etc/letsencrypt/live/j12d202.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/j12d202.p.ssafy.io/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location @forbidden {
        return 302 https://$host/error/permission-denied;
    }
    location @notfound {
        return 302 https://$host/error/not-found;
    }
```

```
    location @bad_gateway {
        return 302 https://$host/error/bad-gateway;
    }

    location / {
        proxy_pass http://frontend;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }

    location /api/ {
        proxy_pass http://backend;
        proxy_http_version 1.1;
        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;

        proxy_read_timeout 3600;
        proxy_cache off;

        proxy_intercept_errors on;
        error_page 403 = @forbidden;
        error_page 502 = @bad_gateway;
    }

    location /api/notification/ {
        proxy_pass http://backend;
        proxy_http_version 1.1;
        proxy_set_header Connection "";
        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;

        # SSE에 중요한 설정
        proxy_buffering off;
        proxy_cache off;
```

```
        proxy_read_timeout 86400s;
        proxy_send_timeout 86400s;

        # chunked transfer 활성화
        proxy_intercept_errors on;
        error_page 403 = @forbidden;
        error_page 502 = @bad_gateway;
        chunked_transfer_encoding on;
    }

    location /ws {
        proxy_pass http://backend;
        proxy_http_version 1.1;
        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;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_read_timeout 86400;    # 24시간 타임아웃 (필요에 따라 조정)
        proxy_send_timeout 86400;    # 24시간 타임아웃 (필요에 따라 조정)

        proxy_intercept_errors on;

        error_page 403 = @forbidden;
        error_page 502 = @bad_gateway;
    }

    location /fastapi/ {
        proxy_pass http://trading-api:8000;
        proxy_http_version 1.1;
        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;

        # 모든 메서드 허용 설정
        proxy_method $request_method;
```

```
        proxy_intercept_errors on;
        error_page 403 = @forbidden;
        error_page 502 = @bad_gateway;
    }

    location /jenkins {
        proxy_pass http://jenkins:8080/jenkins/;
        proxy_http_version 1.1;
        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;

        # Jenkins 관련 추가 설정
        proxy_set_header X-Jenkins-Context "/jenkins";
        proxy_redirect http:// https://;

        proxy_intercept_errors on;
        error_page 404 = @notfound;
        error_page 502 = @bad_gateway;
    }

    location /metrics {
        proxy_pass http://backend/actuator/prometheus;
        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;
        proxy_redirect off;
    }

    # Prometheus 모니터링
    location /prometheus/ {
        proxy_pass http://prometheus:9090/;
        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;
        proxy_redirect / /prometheus/;

        proxy_intercept_errors on;
        error_page 404 = @notfound;
        error_page 502 = @bad_gateway;
    }

    # Grafana 모니터링
    location /grafana/ {
        proxy_pass http://grafana:3000/;
        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;
        proxy_set_header X-Forwarded-Prefix /grafana;
        proxy_redirect off;

        proxy_intercept_errors on;
        error_page 502 = @bad_gateway;
    }
    location /loki/ {
        proxy_pass http://loki:3100/;
        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;

        proxy_intercept_errors on;
        error_page 404 = @notfound;
        error_page 502 = @bad_gateway;
    }

    add_header X-Content-Type-Options "nosniff" always;
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-XSS-Protection "1; mode=block" always;
  }
}
```

9. BackEnd 배포

9.1 app/docker-compose.yml 작성

```
#infra/docker/app/docker-compose.yml
services:

  springboot-app:
  build:
    args:
      - PROFILE=prod
  image: spring-backend
  container_name: spring
  ports:
    - "8081:8081"
  environment:
    - SPRING_PROFILES_ACTIVE=prod
    - PROFILE=prod
    - SPRING_DATASOURCE_URL=jdbc:mysql://mysql:3306/${MYSQL_DATAB
    - SPRING_DATASOURCE_USERNAME=${MYSQL_USER}
    - SPRING_DATASOURCE_PASSWORD=${MYSQL_PASSWORD}
    - SERVER_PORT=8081
  restart: always
  networks:
    - app-network

networks:
  app-network:
    external: true
```

9.2. DockerFile 작성

```
#Dockerfile
FROM amazoncorretto:17

ENV TZ=UTC
```

```
ENV JAVA_OPTS="-Duser.timezone=UTC"

ARG JAR_FILE=./build/libs/ac202-0.0.1-SNAPSHOT.jar
ARG PROFILE
ENV SPRING_PROFILES_ACTIVE=${PROFILE}
WORKDIR /app
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java", "-Dspring.profiles.active=${SPRING_PROFILES_ACTIVE}
```

9.3. Jenkins pipeline을 이용하여 배포

```
pipeline {
    agent any
    stages {
        stage('BE-dev-Checkout') {
            steps {
                echo 'Start Checkout BDL-backend project...'
                git branch: 'develop',
                    credentialsId: 'BDL-backend',
                    url: 'https://lab.ssafy.com/rich-beggar/bdl-backend.git'
                echo 'Checkout finished!'
            }
        }
        stage('BE-dev-Build') {
            steps {
                echo 'Start building BDL-backend project...'
                script {
                    def startTime = System.currentTimeMillis()
                    withCredentials([file(credentialsId: 'application-secret.yml', variab
                        sh """
                            cat "\$SECRET_FILE" > src/main/resources/application-secre
                            cat src/main/resources/application-secret.yml
                        """
                    }
                    sh '''
                        chmod +x ./gradlew
                        ./gradlew clean build -x test
                    '''
```

```
            def endTime = System.currentTimeMillis()
            def duration = (endTime - startTime) / 1000
            echo "🚀 백엔드 빌드 완료: ${duration}초 소요"
        }
        echo 'Build finished!'
    }
}

stage('BE-dev-Build Docker Image') {
    steps {
        script {
            def startTime = System.currentTimeMillis()
            sh "docker build -t spring-backend ."
            def endTime = System.currentTimeMillis()
            def duration = (endTime - startTime) / 1000
            echo "🚀 Docker 이미지 빌드 완료: ${duration}초 소요"
        }
    }
}
stage('Clean Docker Images') {
    steps {
        script {
            sh "docker image prune -f"
        }
    }
}
stage('BE-dev-Deploy') {
    steps {
        script {
            def startTime = System.currentTimeMillis()

            sh "docker stop spring || true"
            sh "docker rm spring || true"

            sh "cd /docker/app && docker compose down springboot-app|| tr
            sh "cd /docker/app && docker compose up -d springboot-app"

            def endTime = System.currentTimeMillis()
```

```
                def duration = (endTime - startTime) / 1000
                echo "🚀 배포 완료: ${duration}초 소요"
            }
        }
    }
}

post {
    success {
        echo '✅ Backend Deployment Successful!'
    }
    failure {
        echo '❌ Backend Deployment Failed.'
    }
}
}
```

10. FrontEnd 배포

10.1. frontend/docker-compose.yml 작성

```
#infra/docker/frontend/docker-compose.yml
services:
  react-app:
    build:
      context: .
    container_name: react
    expose:
      - "80"
    image: react-frontend
    restart: always
    networks:
      - app-network


networks:
```

```
  app-network:
    external: true
```

10.2. DockerFile 작성

```
# Dockerfile
FROM node:22.12.0-alpine AS builder

WORKDIR /app

RUN npm install -g pnpm
COPY package.json pnpm-lock.yaml* ./
RUN pnpm install --frozen-lockfile

COPY . .
RUN pnpm run build

# 프로덕션 단계
FROM nginx:alpine

# Nginx 설정 파일 복사
COPY nginx.conf /etc/nginx/conf.d/default.conf

# 빌드된 파일을 Nginx 서버로 복사
COPY --from=builder /app/dist /usr/share/nginx/html

EXPOSE 80

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

10.3. Nginx.conf 작성

```
#nginx.conf
server {
    listen 80;
    location / {
        root /usr/share/nginx/html;
```

```
        try_files $uri $uri/ /index.html;
    }
}
```

10.4. Jenkins pipeline를 이용하여배포

```
pipeline {
    agent any

    stages {
        stage('FE-dev-Checkout') {
            steps {
                echo 'Start Checkout React frontend project...'
                git branch: 'develop',
                    credentialsId: 'BDL-frontend',
                    url: 'https://lab.ssafy.com/rich-beggar/bdl-frontend.git'
                echo 'Checkout finished!'
            }
        }

        stage('FE-dev-Build Docker Image') {
            steps {
                script {
                    def startTime = System.currentTimeMillis()
                    sh "docker build -t react-frontend ."
                    def endTime = System.currentTimeMillis()
                    def duration = (endTime - startTime) / 1000
                    echo "🚀 Docker 이미지 빌드 완료: ${duration}초 소요"
                }
            }
        }
        stage('Clean Docker Images') {
            steps {
                script {
                    sh "docker image prune -f"
                }
            }
        }
```

```
        stage('FE-dev-Deploy') {
            steps {
                script {
                    def startTime = System.currentTimeMillis()

                    sh "docker stop react || true"
                    sh "docker rm react || true"

                    sh "cd /docker/app && docker compose down react-app || true"
                    sh "cd /docker/app && docker compose up -d react-app"

                    def endTime = System.currentTimeMillis()
                    def duration = (endTime - startTime) / 1000
                    echo "🚀 배포 완료: ${duration}초 소요"
                }
            }
        }
    }

    post {
        success {
            echo '✅ Frontend Deployment Successful!'
        }
        failure {
            echo '❌ Frontend Deployment Failed.'
        }
    }
}
```

11. BackEnd (Fast-API) 배포

11.1. docker-compose.yml 작성

```
  fastapi-trading:
```

```
    image: fastapi-trading
    container_name: trading-api
    restart: always
    ports:
      - "8000:8000"
    environment:
      - REDIS_URL=redis://:${REDIS_PASSWORD}@redis:6379
      - DB_HOST=mysql
      - DB_USER=${MYSQL_USER}
      - DB_PASSWORD=${MYSQL_PASSWORD}
      - DB_NAME=${MYSQL_DATABASE}
      - DB_PORT=3306
      - SPRING_SERVER_URL=http://springboot-app:8081
    networks:
      - app-network

  celery-worker:
    image: fastapi-trading
    container_name: trading-celery-worker
    restart: always
    environment:
      - REDIS_URL=redis://:${REDIS_PASSWORD}@redis:6379
      - DB_HOST=mysql
      - DB_USER=${MYSQL_USER}
      - DB_PASSWORD=${MYSQL_PASSWORD}
      - DB_NAME=${MYSQL_DATABASE}
      - DB_PORT=3306
      - SPRING_SERVER_URL=http://springboot-app:8081
    command: celery -A app.celery_worker worker --loglevel=info
    networks:
      - app-network


networks:
  app-network:
    external: true
```

11.2. DockerFile 작성

```
FROM python:3.11-slim

# 작업 디렉토리 설정
WORKDIR /app

# 시스템 패키지 업데이트 및 필요 패키지 설치
RUN apt-get update && apt-get install -y --no-install-recommends \
    gcc \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/*

# 환경 변수 설정
ENV PYTHONDONTWRITEBYTECODE=1 \
    PYTHONUNBUFFERED=1 \
    PYTHONPATH=/app

# 의존성 설치
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# 애플리케이션 복사
COPY . .

# 포트 노출
EXPOSE 8000

# 서버 실행
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]
```

11.3. Jenkins pipeline을 이용하여 배포

```
pipeline {
    agent any
    stages {
        stage('FA-dev-Checkout') {
            steps {
                echo 'Start Checkout FastAPI project...'
```

```
            git branch: 'main',
                credentialsId: 'BDL-backend-fastapi',
                url: 'https://lab.ssafy.com/rich-beggar/backend-fast-api.git'
            echo 'Checkout finished!'
        }
    }

    stage('FA-dev-Build Docker Image') {
        steps {
            script {
                def startTime = System.currentTimeMillis()
                echo 'Building Docker image (includes Python dependencies)...'
                sh "docker build -t fastapi-trading ."
                def endTime = System.currentTimeMillis()
                def duration = (endTime - startTime) / 1000
                echo "🚀 Docker 이미지 빌드 완료: ${duration}초 소요"
            }
        }
    }
    stage('Clean Docker Images') {
        steps {
            script {
                sh "docker image prune -f"
            }
        }
    }
    stage('FA-dev-Deploy') {
        steps {
            script {
                def startTime = System.currentTimeMillis()

                // 기존 FastAPI 컨테이너 중지 및 삭제
                sh "docker stop trading-api || true"
                sh "docker rm trading-api || true"
                sh "docker stop trading-celery-worker || true"
                sh "docker rm trading-celery-worker || true"

                // docker-compose를 사용하여 FastAPI 서비스만 재시작
```

```
                sh "cd /docker/app && docker compose down fastapi-trading cele
                sh "cd /docker/app && docker compose up -d fastapi-trading cele

                def endTime = System.currentTimeMillis()
                def duration = (endTime - startTime) / 1000
                echo "🚀 배포 완료: ${duration}초 소요"
            }
        }
    }
}

post {
    success {
        echo '✅ FastAPI Deployment Successful!'
    }
    failure {
        echo '❌ FastAPI Deployment Failed.'
    }
}
}
```

12. 모니터링 추가 (Grafana, prometheus, loki, promtail)

12.1. docker-compose.yml 작성

```
 #home/ubuntu/docker/monitoring/docker-compose.yml
services:
  prometheus:
    image: prom/prometheus
    container_name: prometheus
    volumes:
      - ./prometheus:/etc/prometheus
    ports:
      - "9090:9090"
    command:
      - "--config.file=/etc/prometheus/prometheus.yml"
```

```
    restart: always
    networks:
      - app-network

  grafana:
    image: grafana/grafana
    container_name: grafana
    ports:
      - "3000:3000"
    volumes:
      - grafana-data:/var/lib/grafana
    environment:
      - GF_SERVER_ROOT_URL=${DOMAIN}/grafana
      - GF_SERVER_SERVE_FROM_SUB_PATH=false
      - GF_SECURITY_ADMIN_PASSWORD=${GRAFANA_ADMIN_PASSWORD} #
    restart: always
    networks:
      - app-network

  node-exporter:
    image: prom/node-exporter
    container_name: node-exporter
    ports:
      - "9100:9100"
    restart: always
    networks:
      - app-network

  mysql-exporter:
    image: prom/mysqld-exporter
    container_name: mysql-exporter
    environment:
      - DATA_SOURCE_NAME=root:ssafyD202!@tcp(mysql:3306)/
    command:
      - "--mysqld.username=root:ssafyD202!"
      - "--mysqld.address=mysql:3306"
      - "--collect.global_status"
      - "--collect.global_variables"
```

```
      - "--collect.perf_schema.eventsstatements"
      - "--collect.info_schema.innodb_metrics"
      - "--collect.info_schema.innodb_tablespaces"
    ports:
      - "9104:9104"
    restart: always
    networks:
      - app-network

  loki:
    image: grafana/loki:latest
    container_name: loki
    ports:
      - "3100:3100"
    command: -config.file=/etc/loki/local-config.yml
    volumes:
      - ./loki:/etc/loki
    restart: always
    networks:
      - app-network

  promtail:
    image: grafana/promtail:latest
    container_name: promtail
    volumes:
      - /var/log:/var/log
      - ./promtail:/etc/promtail
      - /var/run/docker.sock:/var/run/docker.sock
      - /var/lib/docker/containers:/var/lib/docker/containers
    command: -config.file=/etc/promtail/config.yml
    restart: always
    networks:
      - app-network

networks:
  app-network:
    external: true
```

```
volumes:
  grafana-data:
```

12.1. loki/locla-config.yml

```
auth_enabled: false

server:
  http_listen_port: 3100

common:
  path_prefix: /tmp/loki

ingester:
  lifecycler:
    address: 127.0.0.1
    ring:
      kvstore:
        store: inmemory
      replication_factor: 1
    final_sleep: 0s
  chunk_idle_period: 5m
  chunk_retain_period: 30s

schema_config:
  configs:
    - from: 2020-10-24
      store: tsdb
      object_store: filesystem
      schema: v13
      index:
        prefix: index_
        period: 24h

storage_config:
  tsdb_shipper:
    active_index_directory: /tmp/loki/tsdb-shipper-active
    cache_location: /tmp/loki/tsdb-shipper-cache
```

```
    cache_ttl: 24h
  filesystem:
    directory: /tmp/loki/chunks

compactor:
  working_directory: /tmp/loki/compactor

limits_config:
  reject_old_samples: true
  reject_old_samples_max_age: 168h

chunk_store_config:
  chunk_cache_config:
    embedded_cache:
      enabled: true
      max_size_mb: 100

table_manager:
  retention_deletes_enabled: true
  retention_period: 168h
```

12.2. prometheus/prometheus.yml

```
#prometheus/prometheus.yml
global:
  scrape_interval: 15s  # 15초마다 메트릭 수집

scrape_configs:
  - job_name: 'spring-exporter' # Springboot 데이터 가져오기
    metrics_path: '/actuator/prometheus'
    static_configs:
      - targets: ['spring:8081'] # spring-app metrics를 통해 수집
  - job_name: 'node-exporter' # node-exporter 데이터 가져오기
    static_configs:
      - targets: ['node-exporter:9100'] # node-exporter:9100을 통해 수집
  - job_name: 'mysqld-exporter'
```

```
    static_configs:
      - targets: ['mysql-exporter:9104']
```

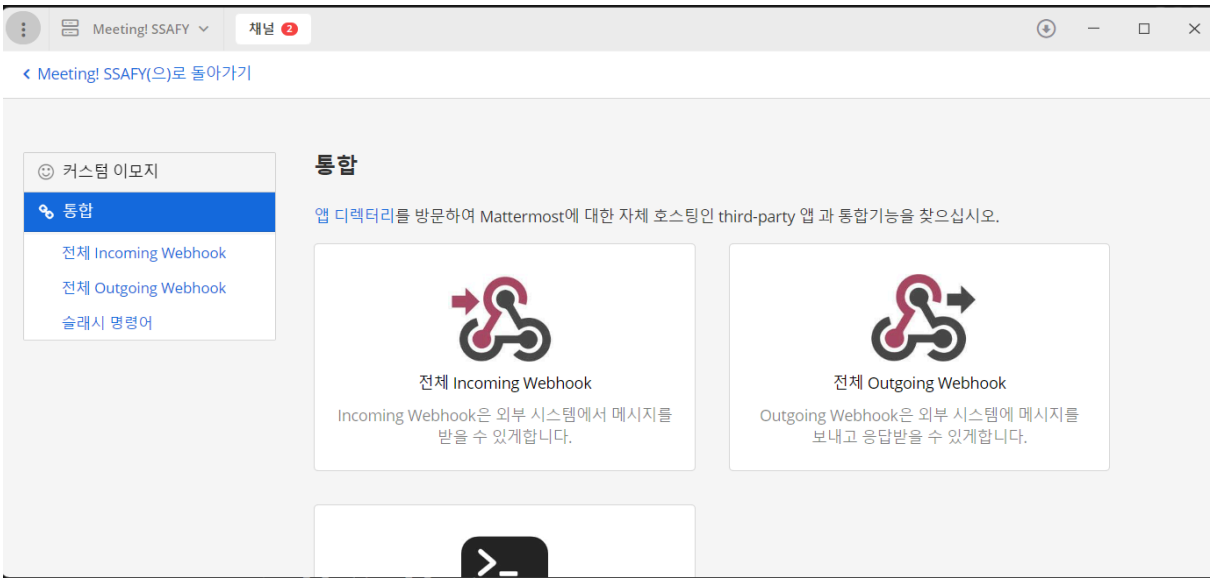
12.3. promtail/config.yml

```
server:
  http_listen_port: 9080
  grpc_listen_port: 0

positions:
  filename: /tmp/positions.yaml

clients:
  - url: http://loki:3100/loki/api/v1/push

scrape_configs:
  - job_name: system
    static_configs:
      - targets:
          - localhost
        labels:
          job: varlogs
          __path__: /var/log/*log
  - job_name: docker
    docker_sd_configs:
        - host: unix:///var/run/docker.sock
          refresh_interval: 5s
    relabel_configs:
      - source_labels: [ '__meta_docker_container_name' ]
        regex: '.*(spring|springboot-app).*'  # Spring 컨테이너만 수집
        action: keep  # 매치되는 컨테이너만 유지
      - source_labels: [ '__meta_docker_container_name' ]
        regex: '/(.*)'
        target_label: 'container'
      - source_labels: [ '__meta_docker_container_name' ]
        regex: '.*'
        replacement: 'app-spring'  # app- 접두사를 붙여서 대시보드 필터와 일치하게 
        target_label: 'compose_service'
```

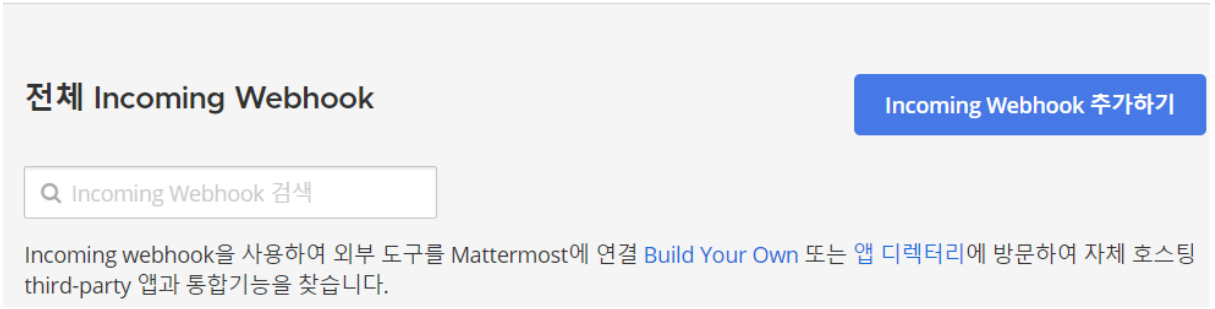
```
    - source_labels: [ '__meta_docker_container_log_stream' ]
      target_label: 'stream'
```

MatterMost Webhook

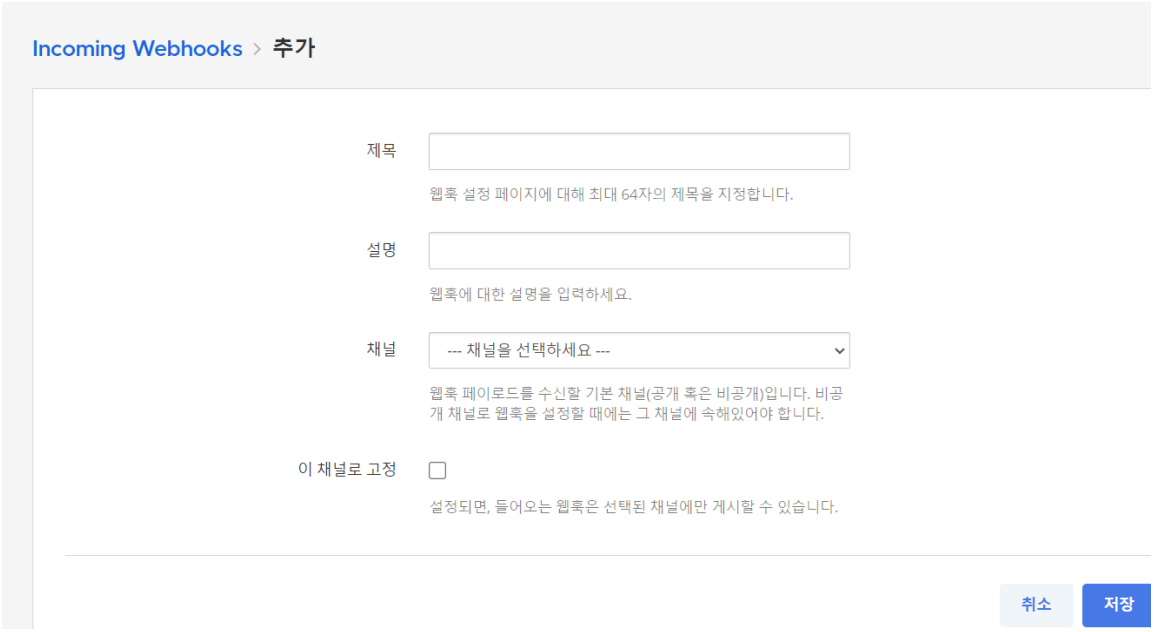
1. mattermost → 목록 → 통합



2. 전체 incoming Webhook 클릭



3. 내용 넣기



4. application-secret.yml에 추가

```
notification:
  mattermost:
    webhookUrl: "https://meeting.ssafy.com/hooks/uu6gtwrp*******ww63o"
```