포팅 메뉴얼

환경변수(**application-secret.yml)**
빌드 배포 메뉴얼
    1. 도커설치
        1.1. Docker 레포지토리 설정
        1.2. 레포지토리 추가
        1.3. Docker 패키지 설치
        1.4. Docker 설치 확인 및 권한 설정
        1.5 Docker 네트워크 생성
    2. Jenkins 설치
        2.1. 호스트 특정 디렉토리에 마운트
        2.2. Jenkins Docker 컨테이너에 설치
        2.3. Jenkins 환경설정
        2.4. config 보안 설정 확인
        2.5. Jenkins 초기 설정
        2.6. Jenkins내 docker명령어 실행
        2.7. Credentials 저장
    3. Jenkins, Gitlab 연동하기
        3.1. Jenkins plugin 설치
        3.2. Credential 등록
        3.3. Gitlab 연결하기
        3.4. **Pipeline 생성 및 Webhook 연결**
    4. MySQL, Redis 컨테이너 설치
        4.1 docker-compose.yml 작성
        4.2. docker 명령어 실행
        4.3 컨테이너 접속 후 권한 부여
    5. NGINX 컨테이너 설치
        5.1 docker-compose.yml 작성
        5.2 CertBot https 인증서 발급
        5.3 Nginx 작성
    6. BackEnd 배포
        6.1 app/docker-compose.yml 작성
        6.2. DockerFile 작성
        6.3. Jenkins pipeline을 이용하여 배포
    7. 모니터링 (Grafana, prometheus, loki, promtail)
        7.1. docker-compose.yml 작성
        7.1. loki/locla-config.yml
        7.2. prometheus/prometheus.yml
        7.3. promtail/config.yml

8. ELK (Filebeat, Logstash, Elasticsearch, Kibana)
8.1 docker-compose.yml 작성
8.2 Dockerfile
8.3 filebeat.yml
8.4 /logstash/pipeline/logstash.conf
9. Kafka + Zookeeper
9.1 docker-compose.yml
MatterMost Webhook
1. Mattermost에 webhook 추가하기
2. AWS Lambda 함수 만들기
3. Gitlab에 webhook 추가하기

환경변수(application-secret.yml)

```
spring:
  config:
    activate:
      on-profile: secret

  datasource:
    url: jdbc:mysql://k12d102.p.ssafy.io/:3306/TAKEN
    username:
    password:
    driver-class-name: com.mysql.cj.jdbc.Driver
  data:
    redis:
      password:

jwt:
  secret:
  access-token-expiration: 86400 # 1일
  refresh-token-expiration: 2592000 # 30일 (86400 * 30)
  token-prefix: "Bearer "
  header-string: "Authorization"

springdoc:
  swagger-ui:
    path: /api/taken/swagger
```

```
    url: /api/v3/api-docs
  api-docs:
    path: /api/v3/api-docs

cloud:
  aws:
    credentials:
      access-key:
      secret-key:
    region:
      static: ap-northeast-2
      auto: false
    s3:
      bucket: boda-taken-bucket
    stack:
      auto: false

firebase:
 service-account-key-path: /app/fcm/boda.json

kakao:
 client-id:  # 카카오 REST API 키
 redirect-uri: "k12d102.p.ssafy.io/api/oauth/kakao"  # 카카오 로그인 완료 후 리다

kakao-map:
  api-key:
```

빌드 배포 메뉴얼

1. 도커설치

1.1. 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
```

1.2. 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
```

1.3. Docker 패키지 설치

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

1.4. 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 do
```

1.5 Docker 네트워크 생성

```
docker network create taken-net
```

2. Jenkins 설치

2.1. 호스트 특정 디렉토리에 마운트

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

2.2. Jenkins Docker 컨테이너에 설치

```
sudo docker run -d \
    --network taken-net \
  -v /home/ubuntu/jenkins-data:/var/jenkins_home \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v /home/ubuntu/docker/proxy:/proxy \
  -v /home/ubuntu/docker/app:/docker/app \
  -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:latest
```

초기 비밀번호 확인 `docker logs jenkins`

2.3. 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
```

2.4. config 보안 설정 확인

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

2.5. Jenkins 초기 설정

2.6. 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
```

2.7. Credentials 저장

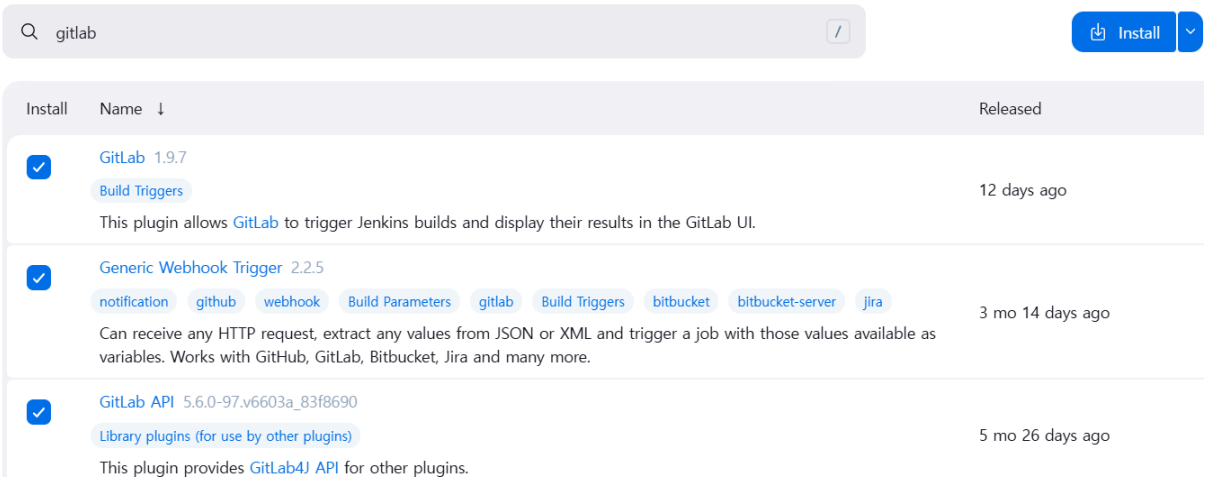
1. .env 작성

```
MYSQL_ROOT_PASSWORD=시크릿키
MYSQL_DATABASE=TAKEN
MYSQL_USER=TAKEN
MYSQL_PASSWORD=시크릿키
```

3. Jenkins, Gitlab 연동하기

3.1. Jenkins plugin 설치

Jenkins관리 → Plugins 클릭



3.2. Credential 등록

Credentials

T	P	Store ↓	Domain	ID	Name
		System	(global)	GITLAB_TOKEN	GitLab API token (gitlab access token)
		System	(global)	GITLAB_LOGIN	seon7129@naver.com/******
		System	(global)	GITLAB_ACCESS_TOKEN	seon7129@naver.com/******
		System	(global)	application-secret.yml	application-secret.yml

Stores scoped to Jenkins

P	Store ↓	Domains
	System	(global)

3.3. Gitlab 연결하기

Gitlab 계정 - Settings - Access Tokens 발급

→ 프로젝트 토큰 발급이 아니라 개인 계정 토큰 발급 필수 !!

Access Token

Token name	Scopes	Created	Last Used	Expires	Role	Action
taken	api, read_api, create_runner, manage_runner, read_repository, write_repository	Apr 29, 2025	2 weeks ago	in 2 weeks	Maintainer	

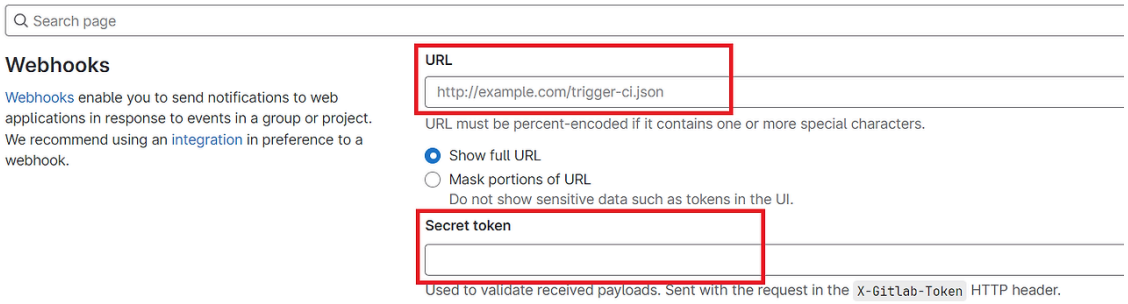
3.4. Pipeline 생성 및 Webhook 연결

jenkins : 새로운 item → pipeline

- Generate 버튼을 클릭 후
  - jenkins secret token 생성

GitLab : 프로젝트 → setting → webhook

- URL과 jenkins Secret token 입력



4. MySQL, Redis 컨테이너 설치

4.1 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; doı
    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
```/docker/db/docker-compose.yml
services:
  mysql:
    image: mysql:8
    container_name: mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD:
      MYSQL_DATABASE: Taken
      MYSQL_USER: Taken
      MYSQL_PASSWORD:
    ports:
      - "13306:3306"
    volumes:
      - ./mysql/init.sql:/docker-entrypoint-initdb.d/init.sql
      - mysql_data:/var/lib/mysql
      - ./mysql/logs:/var/log/mysql
    command:
      - --character-set-server=utf8mb4
      - --collation-server=utf8mb4_unicode_ci
      - --slow_query_log=1
      - --slow_query_log_file=/var/log/mysql/mysql-slow.log
      - --long_query_time=1
    healthcheck:
      test: ["CMD", "mysqladmin", "ping", "-h", "localhost"]
      interval: 10s
      timeout: 5s
      retries: 5
    networks:
      - taken-net
```
```

```
  redis:
    image: redis:7
    container_name: redis
    ports:
      - "16379:6379"
    volumes:
      - redis_data:/data
    command: redis-server --requirepass '[비밀번호]' --appendonly yes
    networks:
      - taken-net

volumes:
  mysql_data:
  redis_data:

networks:
  taken-net:
    external: true
```

4.2. docker 명령어 실행

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

docker compose up -d
```

4.3 컨테이너 접속 후 권한 부여

```
#MYSQL 컨테이너 접속
docker exec -it mysql bash

#MYSQL에 root로 로그인
mysql -u root -p

#password 입력
```

```
#권한 부여 명령어 실행
USE Taken;

CREATE USER 'TAKEN'@'%' IDENTIFIED BY '';
GRANT ALL PRIVILEGES ON TAKEN.* TO 'TAKEN'@'%';
FLUSH PRIVILEGES;

# 로그 수집용
CREATE USER 'exporter_user'@'%' IDENTIFIED BY '';
GRANT PROCESS, REPLICATION CLIENT, SELECT ON *.* TO 'exporter_user'@
FLUSH PRIVILEGES;

#확인하기
SHOW GRANTS FOR 'TAKEN'@'%';

#나가기
exit;
```

5. NGINX 컨테이너 설치

5.1 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:
      - taken-net
    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:
      - taken-net

networks:
  taken-net:
    external: true
```

5.2 CertBot https 인증서 발급

5.2.1 nginx.conf 작성

추후 변경 예정 기본 코드 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;
    }
}
```

5.2.2 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
```

```
sudo docker run --rm \
  -v /home/ubuntu/docker/proxy/data/certbot/conf:/etc/letsencrypt \
  -v /home/ubuntu/docker/proxy/data/certbot/www:/var/www/certbot \
  certbot/certbot certonly --webroot \
  -w /var/www/certbot \
  -d k12d102.p.ssafy.io \
  --agree-tos \
  --email seon7129@naver.com \
  --force-renewal
```

5.2.4 인증서 발급이 성공되면 SSL pem 파일 작성

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

5.3 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;
    }

    server {
        listen 80;
        listen [::]:80;
        server_name k12d102.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 k12d102.p.ssafy.io;
        server_tokens off;

        ssl_certificate /etc/letsencrypt/live/k12d102.p.ssafy.io/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/k12d102.p.ssafy.io/privkey.pem;
        include /etc/letsencrypt/options-ssl-nginx.conf;
```

```
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    # 너무 긴 URI 요청 처리를 위한 named location
    location @uri_too_large {
        return 302 https://$host/error/too-large;
    }

    # URI 길이 체크 (server 블록 시작 부분에 배치)
    if ($request_uri ~* "^.{500,}$") {
        return 414;
    }

    # 에러 페이지 설정 (다른 error_page 설정과 함께 배치)
    error_page 414 = @uri_too_large;

    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 /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;

    }
```

```
    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;
    }

    # swagger 슬래시 없는 요청 리디렉션
    location = /api/taken/swagger {
            return 301 /api/taken/swagger/;
    }

    # swagger 진입점에서 실제 Swagger UI index.html로 바로 이동
            location = /api/taken/swagger/ {
                return 301 /api/taken/swagger-ui/index.html;
            }

    location /api/taken/swagger/ {
            proxy_pass http://backend/api/taken/swagger/;
            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;
    }

    location /api/taken/swagger-ui/ {
                rewrite ^/api/taken/swagger-ui/(.*)$ /api/taken/swagger-ui/$1 bre
```

```
            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;
        }


        location /api/v3/api-docs/ {
            rewrite ^/api/v3/api-docs/(.*)$ /api/v3/api-docs break;
            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;
        }

    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 대시보드
    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 대시보드
    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;
    }


    # Kibana 대시보드
        location = /kibana {
            return 301 /kibana/;
        }

        location /kibana/ {
            proxy_pass http://kibana:5601/;
```

```
            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;

            proxy_redirect off;
            proxy_intercept_errors on;
            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;
  }
}
```

6. BackEnd 배포

6.1 app/docker-compose.yml 작성

```
# /home/ubuntu/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/TAKEN?useSSL=f
    - SPRING_DATASOURCE_USERNAME=
    - SPRING_DATASOURCE_PASSWORD=
    - SERVER_PORT=8081
  volumes:
    - /home/ubuntu/app-logs:/var/log/spring  # Spring 로그 파일 저장용 볼륨 추가
    - /home/ubuntu/docker/app/fcm/boda.json:/app/fcm/boda.json
  restart: always
  networks:
    - taken-net

networks:
  taken-net:
    external: true
```

6.2. DockerFile 작성

```
FROM amazoncorretto:17

ENV SPRING_PROFILES_ACTIVE=prod

WORKDIR /app
COPY ./build/libs/ac102-0.0.1-SNAPSHOT.jar app.jar

CMD ["sh", "-c", "java -Dspring.profiles.active=$SPRING_PROFILES_ACTIVE -
```

6.3. Jenkins pipeline을 이용하여 배포

```
pipeline {
    agent any

    stages {
        stage('BE-dev-Checkout') {
            steps {
                echo 'Start Checkout Taken-backend project...'
                git branch: 'develop',
                    credentialsId: 'GITLAB_LOGIN',
```

```
            url: 'https://lab.ssafy.com/taken/taken_backend.git'
        echo 'Checkout finished!'
    }
}

stage('BE-dev-Build') {
    steps {
        echo 'Start building taken_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
                echo "빌드 후 파일 확인:"
                ls -al build/libs
            '''

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


// stage('SonarQube Analysis') {
//     steps {
//         echo '📊 SonarQube 코드 분석 시작'
//         withSonarQubeEnv('SonarQube') {
```

```
//             sh './gradlew sonarqube'
//         }
//     }
// }


stage('BE-dev-Build Docker Image') {
    steps {
        script {
            def startTime = System.currentTimeMillis()

            // 빌드된 jar 파일명을 변수로 추출 (가장 최근 생성된 파일)
            def jarName = sh(
                script: "ls -t build/libs/*.jar | grep -v plain | head -n 1",
                returnStdout: true
            ).trim()

            echo "📦 사용할 JAR 파일: ${jarName}"

            // Dockerfile이 루트에 있고, COPY 경로를 jarName으로 전달
            sh """
                cp ${jarName} app.jar
                docker build -t spring-backend .
                rm -f app.jar
            """

            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 "docker compose -f /home/ubuntu/docker/app/docker-comp
            // sh "docker compose -f /home/ubuntu/docker/app/docker-comp
            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.'
      }
    }
}
```

7. 모니터링 (Grafana, prometheus, loki, promtail)

7.1. docker-compose.yml 작성

```
# /home/ubuntu/docker/monitoring/grafana/docker-compose.yml
version: "3.8"

services:
  # 📈 Prometheus
  prometheus:
    image: prom/prometheus
    container_name: prometheus
    volumes:
      - ./prometheus:/etc/prometheus
    ports:
      - "9090:9090"
    command:
      - "--config.file=/etc/prometheus/prometheus.yml"
    restart: always
    networks:
      - taken-net

  # 📊 Grafana
  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:
      - taken-net

  # 📡 node exporter
  node-exporter:
```

```
  image: prom/node-exporter
  container_name: node-exporter
  ports:
    - "9100:9100"
  restart: always
  networks:
    - taken-net

# 🧩 MySQL exporter
mysql-exporter:
  image: prom/mysqld-exporter
  container_name: mysql-exporter
  environment:
    - DATA_SOURCE_NAME=exporter_user:[비밀번호]@tcp(mysql:3306)/
  command:
    - "--mysqld.username=exporter_user:[비밀번호]"
    - "--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:
    - taken-net

loki:
  image: grafana/loki:latest
  container_name: loki
  ports:
    - "3100:3100"
  command: -config.file=/etc/loki/local-config.yml
  volumes:
    - ./loki:/etc/loki
    - ./loki-data:/loki  # ✅ tsdb 필수 경로들 저장소
  restart: always
```

```
    networks:
      - taken-net


  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:
      - taken-net

networks:
  taken-net:
    external: true

volumes:
  grafana-data:
  esdata:
```

7.1. loki/locla-config.yml

```
# /home/ubuntu/docker/monitoring/loki/local-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
```

7.2. prometheus/prometheus.yml

```
# /home/ubuntu/docker/monitoring/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']
```

7.3. promtail/config.yml

```
# /home/ubuntu/docker/monitoring/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'
```

8. ELK (Filebeat, Logstash, Elasticsearch, Kibana)

8.1 docker-compose.yml 작성

```
# /home/ubuntu/docker/monitoring/elk/docker-compose.yml
version: "3.8"

services:
  # Elasticsearch
```

```
elasticsearch:
  build:
    context: ./elasticsearch_with_nori
  container_name: elasticsearch
  environment:
    - discovery.type=single-node
    - xpack.security.enabled=false
    - ES_JAVA_OPTS=-Xms1g -Xmx1g
    - ELASTIC_PASSWORD=[비밀번호]
  ports:
    - "9200:9200"
    - "9300:9300"
  networks:
    - taken-net
  volumes:
    - esdata:/usr/share/elasticsearch/data
  mem_limit: 2g


# Kibana
kibana:
  image: docker.elastic.co/kibana/kibana:8.13.4
  container_name: kibana
  ports:
    - "5601:5601"
  environment:
    - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
        - SERVER_BASEPATH=/kibana
        - SERVER_REWRITEBASEPATH=true
  depends_on:
    - elasticsearch
  networks:
    - taken-net

# Logstash
logstash:
  image: docker.elastic.co/logstash/logstash:8.13.4
  container_name: logstash
```

```
    ports:
      - "5044:5044"  # Beats input
    volumes:
      - /home/ubuntu/docker/monitoring/elk/logstash/pipeline:/usr/share/logsta
    depends_on:
      - elasticsearch
    networks:
      - taken-net

  # Filebeat
  filebeat:
    image: docker.elastic.co/beats/filebeat:8.13.4
    container_name: filebeat
    user: root
    volumes:
      - /home/ubuntu/docker/monitoring/elk/filebeat.yml:/usr/share/filebeat/fileb
      - /home/ubuntu/app-logs:/home/ubuntu/app-logs  # Spring 로그 위치
      - /home/ubuntu/docker/db/mysql/logs:/home/ubuntu/docker/db/mysql/log
    depends_on:
      - logstash
    networks:
      - taken-net

networks:
  taken-net:
    external: true

volumes:
  grafana-data:
  esdata:
```

8.2 Dockerfile

```
ARG ES_VERSION=8.14.0
FROM docker.elastic.co/elasticsearch/elasticsearch:${ES_VERSION}

RUN elasticsearch-plugin install analysis-nori
```

8.3 filebeat.yml

```
# /home/ubuntu/docker/monitoring/filebeat.yml
filebeat.inputs:
  - type: log
    enabled: true
    paths:
      - /home/ubuntu/app-logs/spring.log
    fields:
      log_type: spring_log
    fields_under_root: true

  - type: log
    enabled: true
    paths:
      - /home/ubuntu/docker/db/mysql/logs/mysql-slow.log
    fields:
      log_type: mysql_slow_log
    fields_under_root: true

output.logstash:
  hosts: ["logstash:5044"]

setup.kibana:
  host: "http://kibana:5601"
```

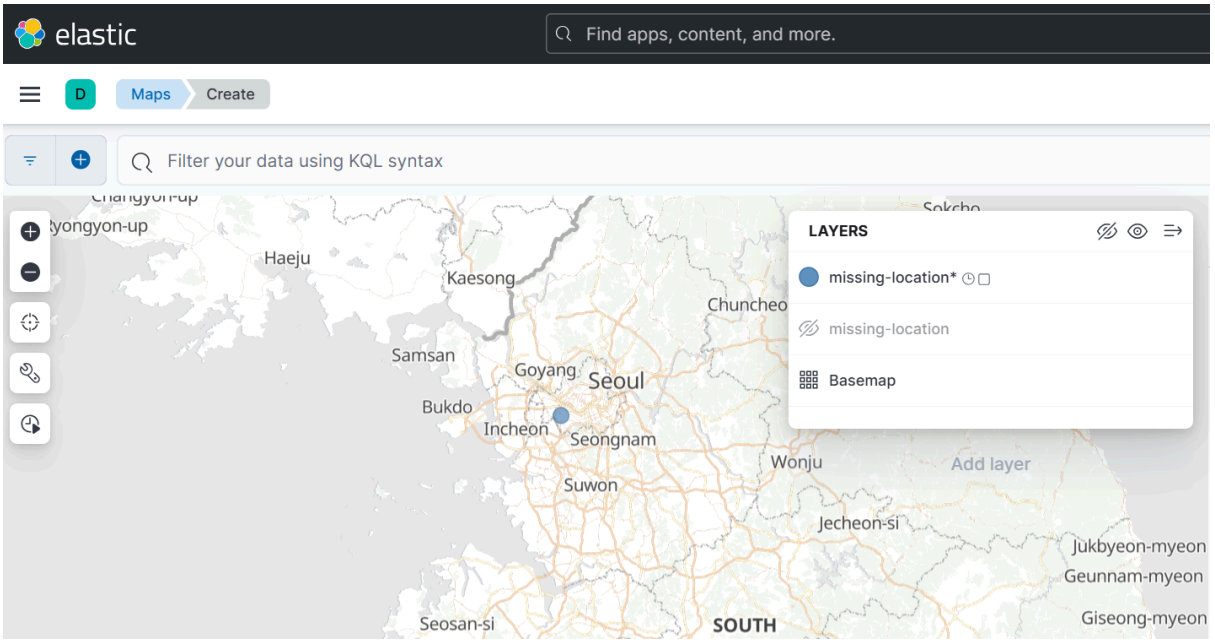
8.4 /logstash/pipeline/logstash.conf

```
# /home/ubuntu/docker/monitoring/logstash/pipeline/logstash.conf
input {
  beats {
    port ⇒ 5044
  }
}

filter {
  if [log_type] == "spring_log" {
    grok {
```

```
      match => {
        "message" => "%{TIMESTAMP_ISO8601:timestamp} %{LOGLEVEL:level}
      }
      tag_on_failure => ["_grokparsefailure_spring"]
    }
    date {
      match => ["timestamp", "ISO8601"]
      target => "@timestamp"
    }
  } else if [log_type] == "mysql_slow_log" {
    grok {
      match => {
        "message" => "^(?<sql>.+)$"
      }
      tag_on_failure => ["_grokparsefailure_mysql"]
    }
    mutate {
      add_field => { "source_type" => "mysql_slow" }
    }
  }
}

output {
  if [log_type] == "spring_log" {
    elasticsearch {
      hosts => ["http://elasticsearch:9200"]
      index => "spring-logs-%{+YYYY.MM.dd}"
    }
  } else if [log_type] == "mysql_slow_log" {
    elasticsearch {
      hosts => ["http://elasticsearch:9200"]
      index => "mysql-slow-%{+YYYY.MM.dd}"
    }
  } else {
    # fallback output for debugging untagged logs
    stdout { codec => rubydebug }
  }
}
```



9. Kafka + Zookeeper

9.1 docker-compose.yml

```
version: '3.8'

services:
  zookeeper:
    image: confluentinc/cp-zookeeper:7.4.0
    container_name: zookeeper
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    networks:
      - taken-net

  kafka:
    image: confluentinc/cp-kafka:7.4.0
    container_name: kafka
```

```
    ports:
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181

      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://k12d102.p.ssafy.io:9092
      KAFKA_LISTENERS: PLAINTEXT://0.0.0.0:9092

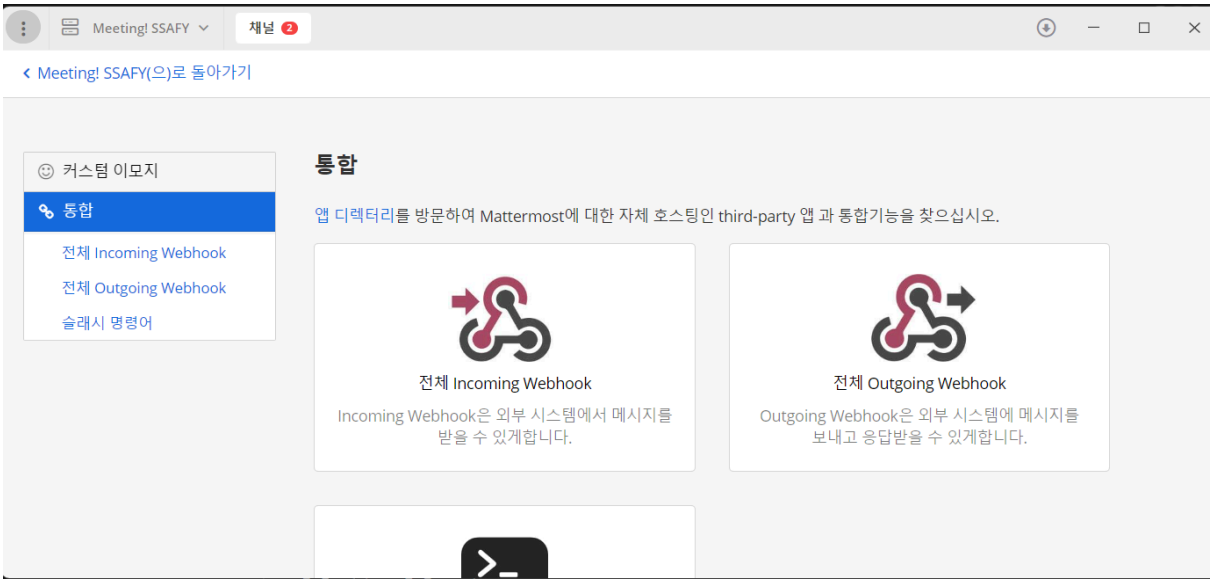
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    depends_on:
      - zookeeper
    networks:
      - taken-net

networks:
  taken-net:
    external: true
```

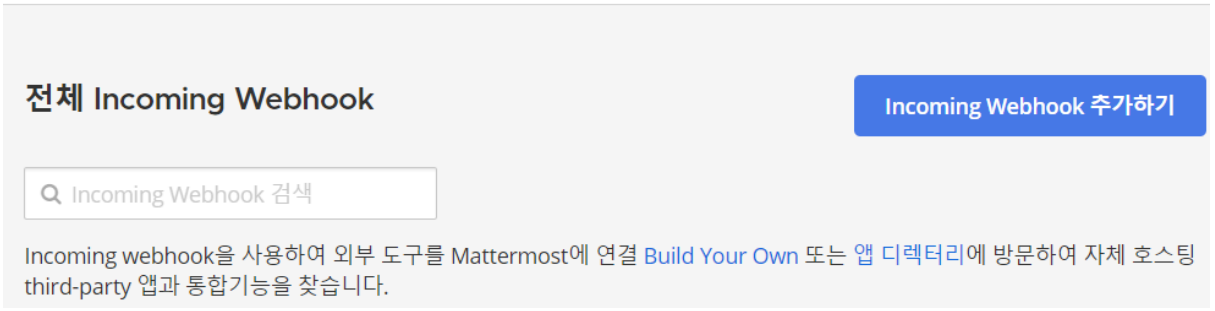
MatterMost Webhook

1. Mattermost에 webhook 추가하기

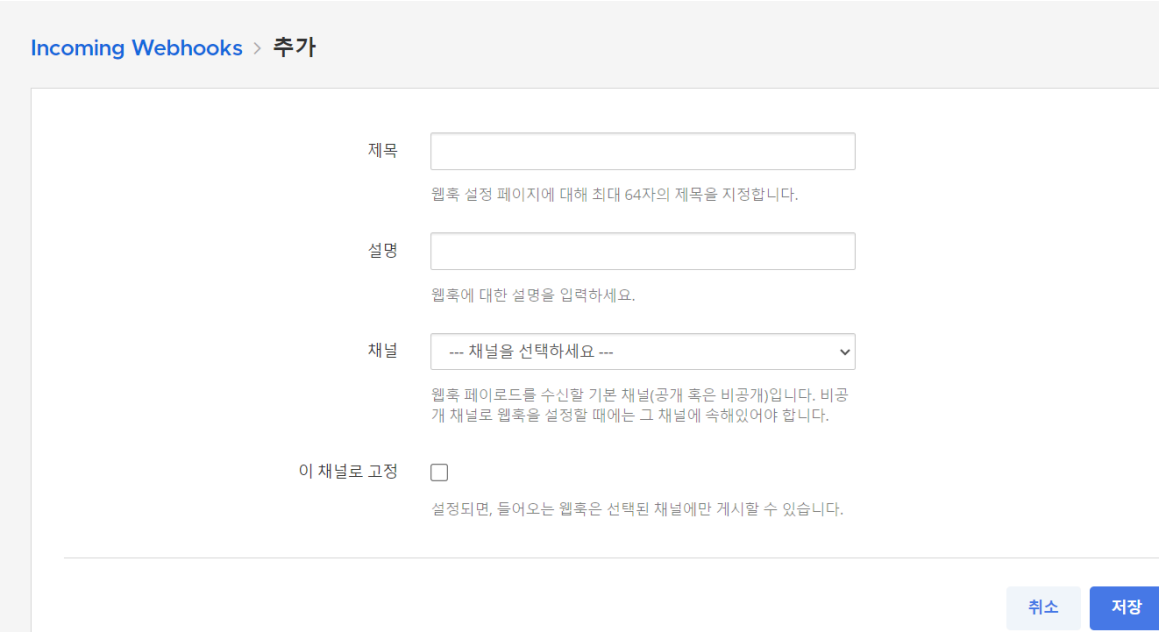
1.1 mattermost → 목록 → 통합



1.2 전체 incoming Webhook 클릭



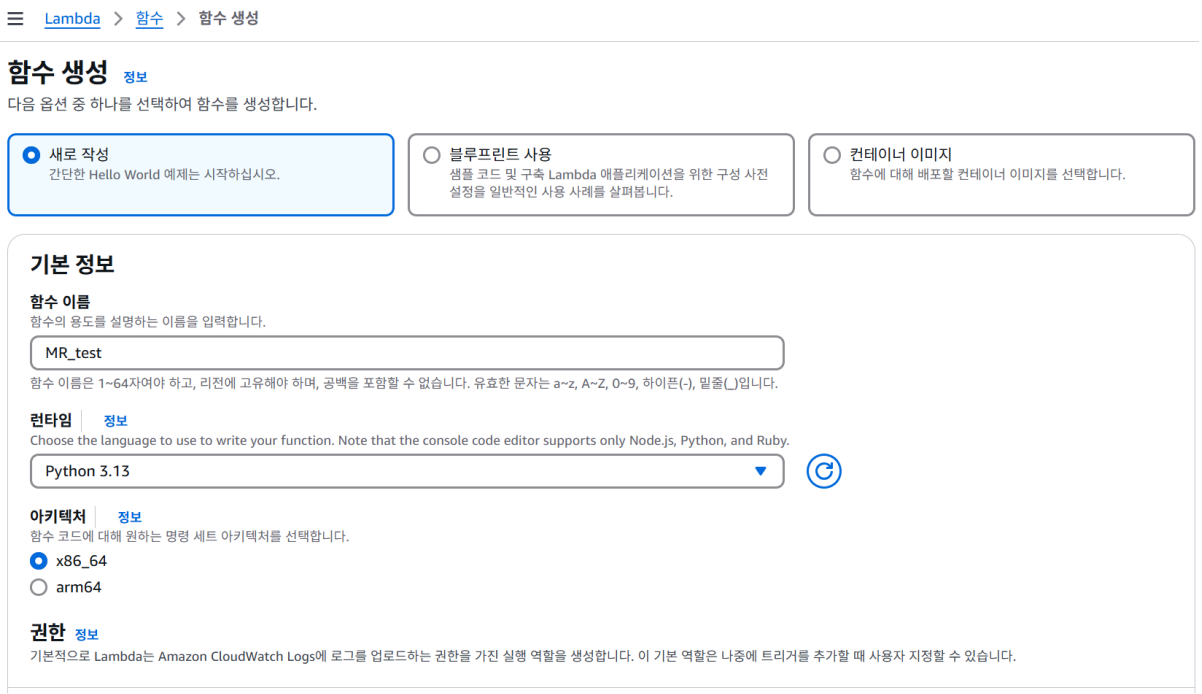
1.3 내용 넣기



1.4 생성 완료

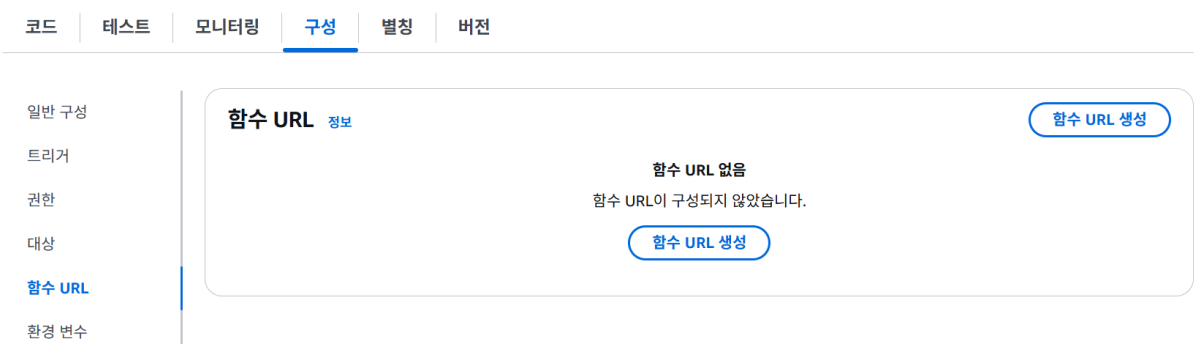
2. AWS Lambda 함수 만들기

2.1 Lambda 함수 생성

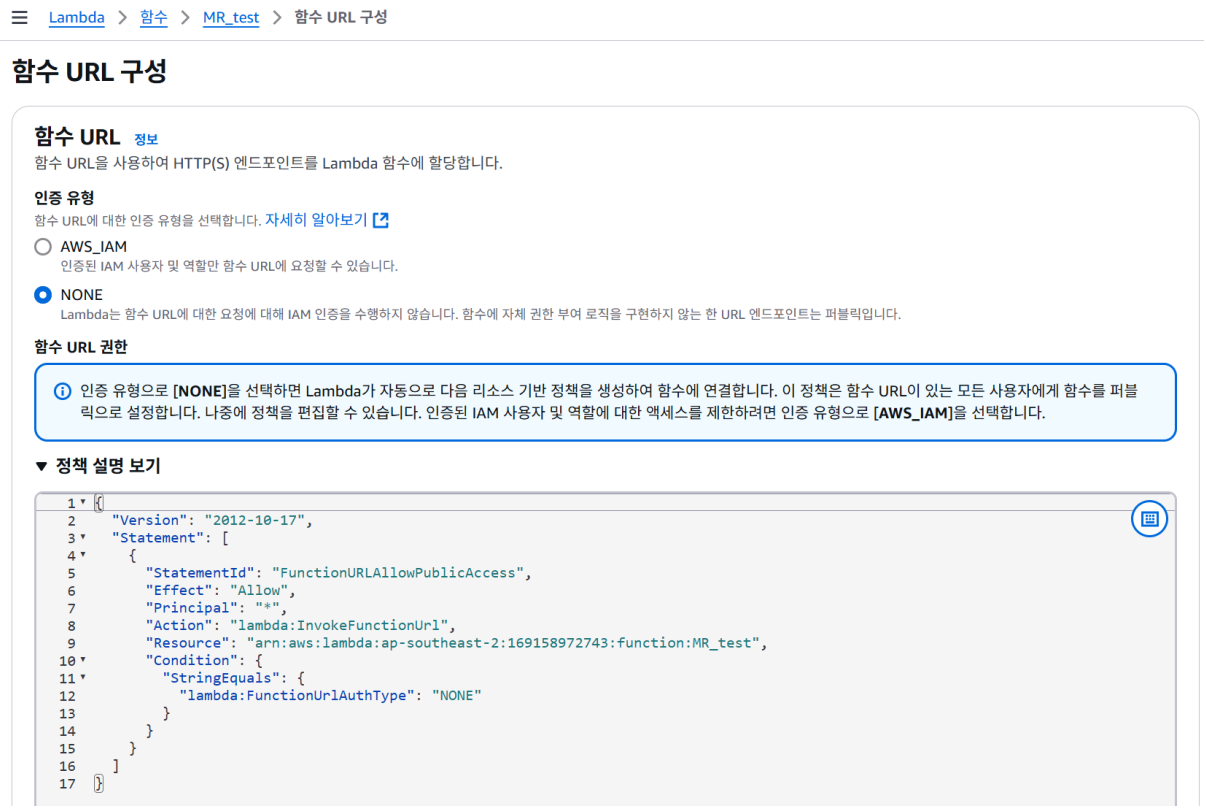


런타임 python 선택

2.2 구성 → 함수 URL 생성

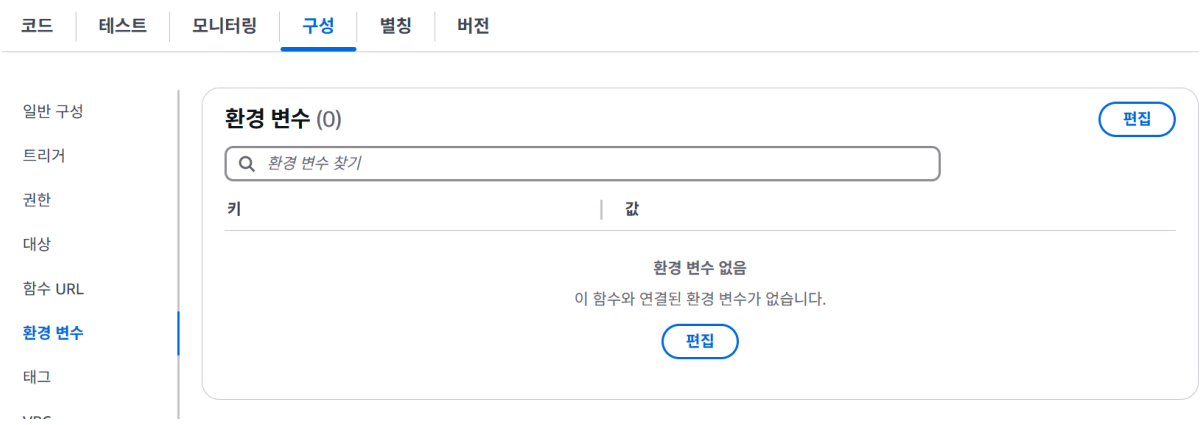


none 선택

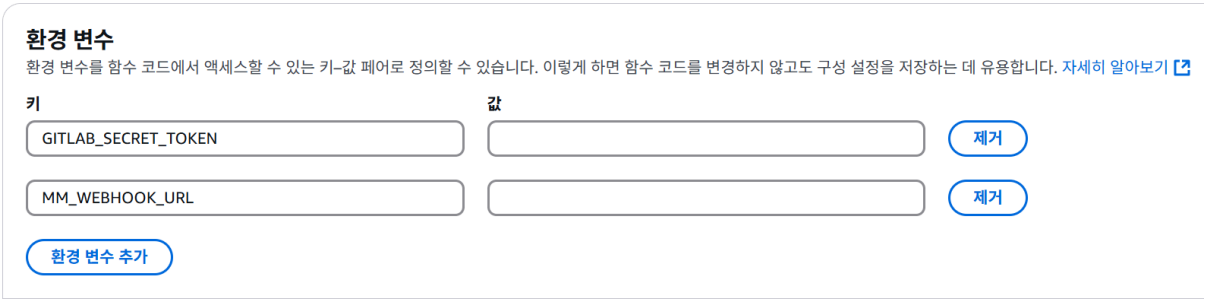


이를 통해 생성된 함수 URL은 Gitlab의 웹훅 설정에 적용

2.3 구성 → 환경변수 편집



환경 변수 편집



MM_WEBHOOK_URL과 GITLAB_SECRET_TOKEN 추가

secret token은 자신이 원하는 값으로. 추후 gitlab webhook 설정에 작성 예정

2.4 코드 작성

```
import json
import os
import urllib.request

# 환경 변수에서 설정 값을 가져옵니다
MM_WEBHOOK_URL = os.environ["MM_WEBHOOK_URL"]  # Mattermost 웹훅
SECRET_TOKEN = os.environ["GITLAB_SECRET_TOKEN"]  # GitLab 시크릿 토큰

# MR 저자 ID로 이름을 찾기 위한 매핑 (실제 구현 필요)
MR_AUTHOR_LOOKUP = {}  # 여기에 author_id: author_name 매핑 추가 필요

def lambda_handler(event, context):
    """
    AWS Lambda 핸들러 함수 - GitLab 웹훅 이벤트를 처리합니다.
```

```
    Args:
        event: Lambda 이벤트 객체 (GibLab 웹훅에서 전달됨)
        context: Lambda 컨텍스트 객체

    Returns:
        응답 객체 (statusCode와 body 포함)
    """
    try:
        print("Received Headers:", json.dumps(event.get("headers", {}), indent=2

        # GitLab 시크릿 토큰 검증
        if not verify_gitlab_token(event):
            return {"statusCode": 403, "body": json.dumps({"error": "Unauthorized

        # 웹훅 이벤트 파싱
        body = json.loads(event["body"])
        event_type = get_event_type(event)

        # 이벤트 타입에 따라 메시지 생성
        message = process_event(event_type, body)

        # Mattermost로 메시지 전송
        send_to_mattermost(message)

        return {"statusCode": 200, "body": json.dumps({"message": "Processed"

    except Exception as e:
        return {"statusCode": 500, "body": json.dumps({"error": str(e)})}

def verify_gitlab_token(event):
    """
    GitLab 웹훅 요청의 시크릿 토큰을 검증합니다.

    Args:
        event: Lambda 이벤트 객체

    Returns:
```

```
        토큰 검증 성공 여부
    """
    received_token = event["headers"].get("X-Gitlab-Token", "") or event["head
    return received_token == SECRET_TOKEN

def get_event_type(event):
    """
    GitLab 이벤트 타입을 추출합니다.

    Args:
        event: Lambda 이벤트 객체

    Returns:
        GitLab 이벤트 타입 문자열
    """
    return event["headers"].get("X-Gitlab-Event", "") or event["headers"].get("

def process_event(event_type, body):
    """
    GitLab 이벤트를 처리하고 적절한 메시지를 생성합니다.

    Args:
        event_type: GitLab 이벤트 타입
        body: 이벤트 본문 데이터

    Returns:
        생성된 메시지
    """
    if event_type == "Merge Request Hook":
        return process_merge_request_event(body)
    elif event_type == "Note Hook":
        return process_note_event(body)
    else:
        raise Exception(f"Unsupported event type: {event_type}")

def extract_mr_data(body, is_note_event=False):
    """
    이벤트 본문에서 MR 관련 데이터를 추출합니다.
```

```
    Args:
        body: 이벤트 본문 데이터
        is_note_event: Note 이벤트인지 여부

    Returns:
        MR 데이터를 포함하는 딕셔너리
    """
    data = {}

    # Note 이벤트와 MR 이벤트의 데이터 구조 차이를 처리
    if is_note_event:
        # Note 이벤트에서는 merge_request 객체에 MR 정보가 있음
        mr_info = body.get("merge_request", {})
        data["title"] = mr_info.get("title", "")
        data["url"] = mr_info.get("url", "")
        data["source_branch"] = mr_info.get("source_branch", "")
        data["target_branch"] = mr_info.get("target_branch", "")
        data["description"] = mr_info.get("description", "")
    else:
        # MR 이벤트에서는 object_attributes에 MR 정보가 있음
        attrs = body["object_attributes"]
        data["title"] = attrs.get("title", "")
        data["url"] = attrs.get("url", "")
        data["source_branch"] = attrs.get("source_branch", "")
        data["target_branch"] = attrs.get("target_branch", "")
        data["description"] = attrs.get("description", "")
        data["action"] = attrs.get("action", "")

    # 공통 데이터
    data["author_name"] = body["user"]["name"]

    return data

def truncate_text(text, max_length=300):
    """
    텍스트가 최대 길이를 초과할 경우 잘라서 반환합니다.
```

```
    Args:
        text: 원본 텍스트
        max_length: 최대 길이

    Returns:
        잘린 텍스트 또는 원본 텍스트
    """
    if len(text) > max_length:
        return text[:max_length-3] + "..."
    return text

def process_merge_request_event(body):
    """
    Merge Request 이벤트를 처리합니다.

    Args:
        body: 이벤트 본문 데이터

    Returns:
        생성된 메시지
    """
    # 데이터 추출
    mr_data = extract_mr_data(body)

    # 액션에 따른 제목 설정
    mm_title = get_mr_action_title(mr_data["action"], mr_data["author_name"])

    if mm_title is None:
        raise Exception(f"Unsupported merge request action: {mr_data['action']}

    # 설명 텍스트 처리
    description = truncate_text(mr_data["description"])

    # 메시지 구성
    message = (
        f"## {mm_title}\n"
        f"- 제목: {mr_data['title']}\n"
        f"- 소스: `{mr_data['source_branch']}`\n"
```

```
        f"- 타겟: `{mr_data['target_branch']}`\n"
    )

    # 설명이 있고 open, reopen, update 액션인 경우에만 설명 추가
    if description and mr_data["action"] in ["open", "reopen", "update"]:
        message += "- 설명:\n```\n" + description + "\n```\n"

    message += f"### [바로가기]({mr_data['url']})"

    return message

def get_mr_action_title(action, mr_author):
    """
    MR 액션에 따른 Mattermost 메시지 제목을 반환합니다.

    Args:
        action: MR 액션 (open, reopen, update, approved, close, merge)
        mr_author: MR 작성자 이름

    Returns:
        포맷된 메시지 제목 또는 None (지원되지 않는 액션일 경우)
    """
    titles = {
        "open": f":star: {mr_author}님의 MR 리뷰해주세요! :star:",
        "reopen": f":star: {mr_author}님의 MR 리뷰해주세요! :star:",
        "update": f":arrows_counterclockwise: {mr_author}님이 MR을 업데이트 했어
        "approved": f":thumbsup: {mr_author}님이 MR을 Approve 했어요! :thumbsu
        "close": f":no_entry_sign: {mr_author}님의 MR이 닫혔어요! :no_entry_sign:",
        "merge": f":white_check_mark: {mr_author}님의 MR이 머지 되었어요! :white
    }

    return titles.get(action)

def process_note_event(body):
    """
    Note 이벤트를 처리합니다.(MR에 남긴 코멘트)

    Args:
```

```
        body: 이벤트 본문 데이터

    Returns:
        생성된 메시지
    """
    noteable_type = body["object_attributes"].get("noteable_type", "")

    if noteable_type == "MergeRequest":
        # 데이터 추출
        mr_data = extract_mr_data(body, is_note_event=True)
        comment_text = body["object_attributes"].get("note", "")

        # 코멘트 내용이 너무 길 경우 요약
        comment_text = truncate_text(comment_text)

        mm_title = f":speech_balloon: {mr_data['title']}에 {mr_data['author_name'

        message = (
            f"## {mm_title}\n"
            f"- 코멘트 작성자: {mr_data['author_name']}\n"
        )

        # 코멘트 내용 추가
        if comment_text:
            message += "- 코멘트:\n```\n" + comment_text + "\n```\n"

        message += f"### [바로가기]({mr_data['url']})"

        return message
    else:
        raise Exception(f"Unsupported note hook type: {noteable_type}")

def send_to_mattermost(message):
    """
    Mattermost 웹훅으로 메시지를 전송합니다.

    Args:
        message: 전송할 메시지 내용
```

```
    Returns:
        HTTP 응답 상태 코드
    """
    payload = json.dumps({
        "text": message
    }).encode("utf-8")  # 페이로드를 바이트로 인코딩

    headers = {"Content-Type": "application/json"}

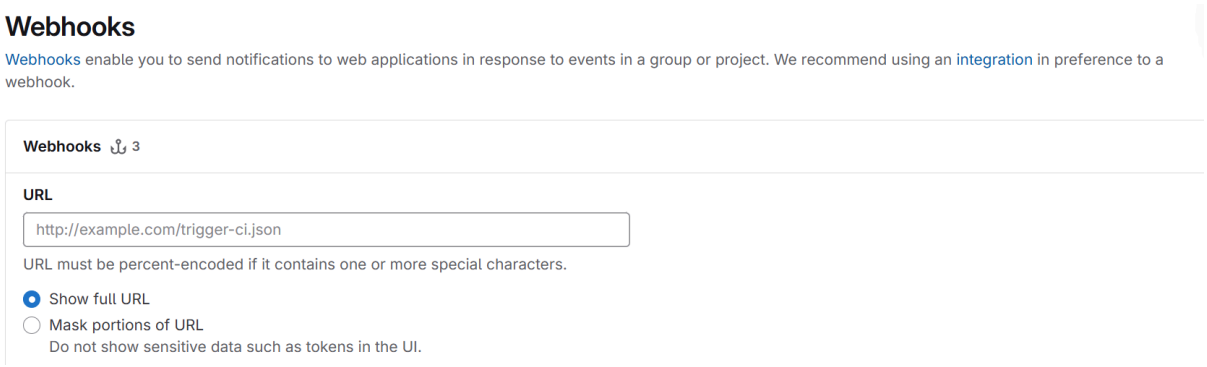
    req = urllib.request.Request(MM_WEBHOOK_URL, data=payload, headers=h
    with urllib.request.urlopen(req) as response:
        return response.status
```

2.5 Deploy로 배포

3. Gitlab에 webhook 추가하기

3.1 gitlab → settings → webhooks → add new webhook

프로젝트에 maintainer 이상의 권한이 있어야 setting이 가능하다



URL에 2.2에서 생성한 함수 URL 작성

secret token에는 Lambda의 환경 변수로 추가한 암호 작성

정유선[구미_1반_D102]팀원 BOT 오후 4:28

👍 정유선님이 MR을 Approve 했어요! 👍

- 제목: S12P31D102-138/✨/위도경도로 도로명주소 조회 구현
- 소스: `S12P31D102-138/change-address`
- 타겟: `develop`

바로가기

정유선[구미_1반_D102]팀원 BOT 오후 4:28

✅ 정유선님의 MR이 머지 되었어요! ✅

- 제목: S12P31D102-138/✨/위도경도로 도로명주소 조회 구현
- 소스: `S12P31D102-138/change-address`
- 타겟: `develop`

바로가기

정유선[구미_1반_D102]팀원 BOT 오후 4:59

⭐ 정지원님의 MR 리뷰해주세요! ⭐

- 제목: S12P31D102-141/✨/도움 요청 및 탐색 종료 시 FCM으로 알림 전송