0. 시스템 환경 정보

1. **운영 체제 (OS):**
    - **배포판:** Ubuntu 20.04.6 LTS (Focal Fossa)
    - **버전 ID:** 20.04
2. **Docker:**
    - **버전:** 27.3.1
    - **빌드 번호:** ce12230
3. **Docker Compose:**
    - **버전:** v2.30.1
4. **Ports**

서비스	호스트 포트	컨테이너 포트
nginx	80	80
nginx	443	443
jenkins	9000	8080
spring	8080	8080
react-native	8081	8081
nextjs	3000	3000
fast-api	8000	8000
postgresql	5444	5432
redis	6379	6379
elasticsearch	9200	9200
logstash	5044	5044
kibana	5601	5601

5. **env 환경변수**

```
POSTGRES_USER=
POSTGRES_PASSWORD=
POSTGRES_DB=
POSTGRES_PORT=
POSTGRES_HOST=
ELASTIC_USERNAME=
ELASTIC_PASSWORD=
OPENAI_API_KEY=
```

```
KIBANA_SYSTEM_PASSWORD=
SECRET_KEY=
```

1. docker 빌드

docker 이미지 목록

IMAGE	TAG
fast-api	latest
spring	latest
nextjs	latest
react-native	latest
jenkins-main	latest
docker.elastic.co/elasticsearch/elasticsearch	8.15.3
docker.elastic.co/kibana/kibana	8.15.3
docker.elastic.co/logstash/logstash	8.15.3
redis	latest
nginx	latest
ramsrib/pgvector	12
certbot/certbot	latest

docker 설치

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt update

sudo apt install docker-ce -y
```

react-native dockerfile

```
# 1. Node.js 이미지 사용
FROM node:16 as builder
```

```
# 2. 작업 디렉토리 설정
WORKDIR /app

# 3. 의존성 설치
COPY package.json package-lock.json ./
RUN npm install

# 4. 소스 코드 복사
COPY . .

# 5. Metro 서버 실행 (빌드 및 개발 환경)
CMD ["npx", "react-native", "start"]
```

nextjs dockerfile

```
# Stage 1: Next.js 애플리케이션 빌드 단계
FROM node:20.15.0 AS builder

# 작업 디렉토리 설정
WORKDIR /app

# package.json과 package-lock.json 파일 복사
COPY package*.json ./

# 의존성 설치
RUN npm install

# 애플리케이션 코드 복사
COPY . .

# Next.js 애플리케이션 빌드
RUN npm run build

# Stage 2: 빌드된 애플리케이션 실행 단계
FROM node:20.15.0

# 빌드 단계에서 생성된 파일 복사
COPY --from=builder /app/.next ./.next
COPY --from=builder /app/public ./public
COPY --from=builder /app/package.json ./package.json

# 프로덕션 환경의 의존성만 설치
RUN npm install --production

# Next.js 애플리케이션 시작
CMD ["npm", "start"]
```

spring dockerfile

```
# 1. Amazon Corretto 17 이미지를 사용
FROM amazoncorretto:17 as builder

# 2. 작업 디렉토리 설정
WORKDIR /app

# 3. 프로젝트의 gradle과 소스 파일 복사
COPY build.gradle settings.gradle gradlew ./
COPY gradle gradle
COPY src src

# 4. 필요한 권한을 부여하고 Gradle을 사용하여 프로젝트 빌드
RUN chmod +x gradlew
RUN ./gradlew build --no-daemon

# 5. 빌드 결과물을 최종 이미지에 복사
FROM amazoncorretto:17
WORKDIR /app
COPY --from=builder /app/build/libs/*.jar app.jar

# 6. 애플리케이션 실행
ENTRYPOINT ["java", "-jar", "app.jar"]
```

fastapi dockerfile

```
# Conda 환경을 제공하는 기본 이미지로 시작
FROM continuumio/miniconda3

# 작업 디렉토리를 /app으로 설정
WORKDIR /app

# PostgreSQL 클라이언트 및 필수 라이브러리 설치
RUN apt-get update && apt-get install -y \
    libpq-dev \
    postgresql-client \
    build-essential && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

# requirements.txt만 별도로 복사하여 캐싱 최적화
COPY backend/requirements.txt /app/backend/requirements.txt

# Conda 환경 생성 및 패키지 설치
RUN conda create -n fastapi_env python=3.10.11 -y && \
    conda run -n fastapi_env pip install --no-cache-dir -r
/app/backend/requirements.txt && \
    conda clean -afy

# SHELL 명령어를 통해 환경 활성화 설정
SHELL ["conda", "run", "-n", "fastapi_env", "/bin/bash", "-c"]
```

```
# PYTHONPATH 설정으로 fast_api를 루트로 추가
ENV PYTHONPATH=/app/backend/fast_api

# 나머지 코드 복사 (변경된 경우에만 캐시 무효화)
COPY backend/ /app/backend
COPY config.py /app/config.py

# FastAPI 애플리케이션 실행 명령어
CMD ["conda", "run", "--no-capture-output", "-n", "fastapi_env", "uvicorn",
"backend.fast_api.main:app", "--host", "0.0.0.0", "--port", "8000", "--log-level",
"debug"]
```

2. docker-compose

docker-compose 설치

```
curl -L "https://github.com/docker/compose/releases/download/$(curl -s
https://api.github.com/repos/docker/compose/releases/latest | grep -oP
'"tag_name": "\K(.*)(?=")')/docker-compose-$(uname -s)-$(uname -m)" -o
/usr/local/bin/docker-compose
```

docker-compose.yml

```
version: '3.8'

services:
  nginx:
    image: nginx:latest
    container_name: nginx
    ports:
      - "80:80"       # HTTP
      - "443:443"     # HTTPS
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf         # nginx.conf 파일 마운트
      - /etc/letsencrypt:/etc/letsencrypt          # 인증서 저장 경로
      - /var/lib/letsencrypt:/var/lib/letsencrypt  # Certbot 관련 데이터 경로
      - /var/www/certbot:/var/www/certbot          # Certbot challenge 파일 경로
    networks:
      - app-network
    depends_on:
      - nextjs
      - spring
      - react-native
      - certbot
      - jenkins

  certbot:
    image: certbot/certbot
```

```
    container_name: certbot
    # 인증서 갱신
    entrypoint: "/bin/sh -c 'trap exit TERM; while :; do certbot renew; sleep 12h
& wait $${!}; done;'"
    volumes:
      - /etc/letsencrypt:/etc/letsencrypt          # 인증서 저장 경로
      - /var/lib/letsencrypt:/var/lib/letsencrypt  # Certbot 관련 데이터 경로
      - /var/www/certbot:/var/www/certbot          # 웹 인증용 파일 저장 경로
    networks:
      - app-network

  jenkins:
    image: jenkins-main:latest
    container_name: jenkins
    ports:
      - "9000:8080"
      - "50000:50000"
    environment:
      - JENKINS_OPTS=--prefix=/jenkins  # Jenkins URL prefix 설정
    volumes:
      - /var/lib/docker/volumes/jenkins_home/_data:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock  # Docker 접근을 위한 소켓
    networks:
      - app-network

  spring:
    image: spring
    container_name: spring
    ports:
      - "8080:8080"
    networks:
      - app-network
    depends_on:
      - postgresql
      - redis

  react-native:
    image: react-native
    container_name: react-native
    ports:
      - "8081:8081"
    networks:
      - app-network

  nextjs:
    image: nextjs:latest
    container_name: nextjs
    ports:
      - "3000:3000"
    environment:
      - NODE_ENV=production
    networks:
      - app-network
```

```
  fast-api:
    image: fast-api:latest
    container_name: fast-api
    ports:
      - "8000:8000"
    networks:
      - app-network
    depends_on:
      - postgresql
      - redis
    environment:
      - POSTGRES_USER=${POSTGRES_USER}
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
      - POSTGRES_DB=${POSTGRES_DB}
      - POSTGRES_PORT=${POSTGRES_PORT}
      - POSTGRES_HOST=${POSTGRES_HOST}
      - OPENAI_API_KEY=${OPENAI_API_KEY}
      - SECRET_KEY=${SECRET_KEY}
      - ALGORITHM=${ALGORITHM}

  postgresql:
    image: ramsrib/pgvector:12
    container_name: postgresql
    environment:
      POSTGRES_USER: ${POSTGRES_USER}
      POSTGRES_PASSWORD: ${POSTGRES_PASSWORD}
      POSTGRES_DB: ${POSTGRES_DB}
    ports:
      - "5444:5432"
    networks:
      - app-network
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: redis:latest
    container_name: redis
    ports:
      - "6379:6379"
    networks:
      - app-network
    volumes:
      - redis_data:/data

  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.15.3
    container_name: elasticsearch
    environment:
      - network.host=0.0.0.0
      - discovery.type=single-node
      - bootstrap.memory_lock=true
      - "ES_JAVA_OPTS=-Xms1g -Xmx1g"
      - xpack.security.enabled=true  # username/password 보안을 활성화
      - ELASTIC_USERNAME=${ELASTIC_USERNAME}
```

```
      - ELASTIC_PASSWORD=${ELASTIC_PASSWORD}
      - logger.org.elasticsearch.transport=DEBUG
      - logger.org.elasticsearch.rest=DEBUG
    ulimits: # 메모리 잠금 해제
      memlock:
        soft: -1
        hard: -1
    ports:
      - "9200:9200"
    volumes:
      - elasticsearch_data:/usr/share/elasticsearch/data
    networks:
      - app-network

  logstash:
    image: docker.elastic.co/logstash/logstash:8.15.3
    container_name: logstash
    environment:
      - xpack.monitoring.elasticsearch.hosts=http://elasticsearch:9200
      - xpack.monitoring.enabled=true # 모니터링 데이터 활성화
      - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
      - ELASTIC_USERNAME=${ELASTIC_USERNAME}
      - ELASTIC_PASSWORD=${ELASTIC_PASSWORD}
    volumes:
      - ./logstash/logstash.conf:/usr/share/logstash/pipeline/logstash.conf
      - ./logstash/postgresql.jar:/usr/share/logstash/postgresql.jar
    ports:
      - "5044:5044"
    networks:
      - app-network
    depends_on:
      - elasticsearch

  kibana:
    image: docker.elastic.co/kibana/kibana:8.15.3
    container_name: kibana
    environment:
      - SERVER_HOST=0.0.0.0
      - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
      - ELASTICSEARCH_USERNAME=kibana_system
      - ELASTICSEARCH_PASSWORD=${KIBANA_SYSTEM_PASSWORD}
      - SERVER_BASEPATH=/kibana
      - SERVER_REWRITEBASEPATH=true
      - xpack.monitoring.ui.enabled=true # logstash 모니터링
    ports:
      - "5601:5601"
    networks:
      - app-network
    depends_on:
      - elasticsearch

networks:
  app-network:
    driver: bridge
```

```
volumes:
  jenkins_home:
  postgres_data:
  redis_data:
  elasticsearch_data:
```

3. nginx

SSL 인증서 발급

- ec2에서 명령어로 최초 발급
- 이후 docker-compose.yml, nginx.conf 설정을 통해 주기적으로 갱신

```
docker-compose run certbot certonly --webroot -w /var/www/certbot -d
k11b105.p.ssafy.io --email your@email.com --agree-tos --non-interactive --config-
dir /etc/letsencrypt --work-dir /var/lib/letsencrypt --logs-dir
/var/log/letsencrypt
```

nginx.conf

```
events {}

http {
    # HTTP 서버 설정: HTTP 요청을 HTTPS로 리디렉션
    server {
        listen 80;
        listen [::]:80;
        server_name k11b105.p.ssafy.io;

        # Let's Encrypt 인증서 갱신을 위한 설정
        location /.well-known/acme-challenge/ {
            root /var/www/certbot;
        }

        # 모든 HTTP 요청을 HTTPS로 리디렉션
        location / {
            return 301 https://$host$request_uri;
        }
    }

    # HTTPS 서버 설정
    server {
        listen 443 ssl;
        listen [::]:443 ssl;
        server_name k11b105.p.ssafy.io;
        client_max_body_size 10M;

        # SSL 인증서 경로 (Let's Encrypt)
```

```
        ssl_certificate /etc/letsencrypt/live/k11b105.p.ssafy.io/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/k11b105.p.ssafy.io/privkey.pem;
        include /etc/letsencrypt/options-ssl-nginx.conf;
        ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

        # 기본 경로 - NextJS 서비스로 프록시
        location / {
            proxy_pass http://nextjs: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;
        }

        # /native 경로 - React Native 서비스로 프록시
        location /native {
            proxy_pass http://react-native:8081;
            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;
        }

        # /wassu 경로 - Spring 서비스로 프록시
        location /wassu {
            proxy_pass http://spring:8080;
            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;

            # SSE 연결 유지를 위한 설정
            proxy_http_version 1.1;  # HTTP/1.1 사용 (SSE는 HTTP/1.1에서 잘 동작)
            proxy_set_header Connection 'keep-alive';  # 연결 유지 설정 (SSE에서 중
요)

            # 타임아웃 설정 (연결이 끊어지지 않도록)
            proxy_read_timeout 3600s;  # 1시간 동안 데이터가 없더라도 연결을 유지
            proxy_send_timeout 3600s;  # 1시간 동안 데이터 전송을 기다림
            send_timeout 3600s;        # 클라이언트로 보내는 데이터의 타임아웃

            # 버퍼링 비활성화 (SSE 실시간 데이터 전송을 위한 설정)
            proxy_buffering off;  # SSE 데이터를 버퍼링하지 않고 바로 전송
            keepalive_timeout 65s;  # 연결이 유지될 시간을 설정 (60초보다 긴 값으로
설정)
        }

        # /jenkins 경로 - Jenkins 서비스로 프록시
        location /jenkins {
            proxy_pass http://jenkins:8080;
            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 http://jenkins:8080/ /jenkins/;
        }

        # /elasticsearch 경로 - Elasticsearch 서비스로 프록시
        location /elasticsearch/ {
            proxy_pass http://elasticsearch:9200/;
            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;
        }

        # /kibana 경로 - Kibana 서비스로 프록시
        location /kibana {
            proxy_pass http://kibana:5601;
            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;
        }

        # /fast_api 경로 - Fast API 서비스로 프록시
        location /fast_api {
            proxy_pass http://fast-api:8000;
            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;
        }

        # /swagger 경로 - Spring Swagger로 프록시
        location /swagger {
            proxy_pass http://spring:8080;
            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;
        }
    }
}
```

4. jenkins

추가 플러그인 설치

- Mattermost Notification
- GitLab
- ssh agent

Mattermost 알림 설정

1. mm 좌측상단 통합으로 이동

2. 전체 Incoming Webhook 선택
3. Incoming Webhook 추가 후 연결할 채널 선택
4. url 및 채널 저장
5. jenkins 관리 -> system -> Global Mattermost Notifier Settings
6. url 및 채널 등록

react-native pipeline

```
pipeline {
    agent any
    environment {
        IP = credentials('server_IP')
    }

    stages {
        // 1단계: Git 클론
        stage('Git Clone') {
            steps {
                git branch: 'dev-fe', credentialsId: 'gitlab-credentials', url:
'https://lab.ssafy.com/s11-final/S11P31B105.git'
            }
            post {
                success { echo 'Repository clone 성공!' }
                failure { echo 'Repository clone 실패!' }
            }
        }

        // 2단계: Docker 이미지 빌드
        stage('Build Docker Image') {
            steps {
                dir('./FrontendApp') {
                    sh 'docker build --no-cache -t react-native .'
                }
                echo 'Docker 이미지 빌드 완료'
            }
            post {
                success { echo '이미지 빌드 성공' }
                failure { echo '이미지 빌드 실패' }
            }
        }

        // 3단계: 기존 컨테이너 중지 및 제거
        stage('Stop and Remove Existing Container') {
            steps {
                echo '기존 React-Native 컨테이너 중지 및 제거 시작'
                sshagent(credentials: ['ec2']) {
                    sh '''
                        ssh -o StrictHostKeyChecking=no -p 2201 ubuntu@$IP "
                            docker-compose stop react-native &&
                            docker-compose rm -f react-native
                        "
                    '''
```

```
                }
                echo '기존 React-Native 컨테이너 중지 및 제거 완료'
            }
            post {
                success { echo '컨테이너 중지 및 제거 성공' }
                failure { echo '컨테이너 중지 및 제거 실패' }
            }
        }

        // 4단계: 새로운 컨테이너 배포
        stage('Deploy New Container') {
            steps {
                echo 'React-Native 서비스 배포 시작'
                sshagent(credentials: ['ec2']) {
                    sh '''
                        ssh -o StrictHostKeyChecking=no -p 2201 ubuntu@$IP "
                            docker-compose up -d --remove-orphans react-native
                        "
                    '''
                }
                echo 'React-Native 서비스 배포 완료'
            }
            post {
                success { echo '배포 성공' }
                failure { echo '배포 실패' }
            }
        }
    }

    post {
        always {
            echo 'Pipeline Execution Complete.'
        }
        success {
            echo 'Pipeline Execution Success.'
            script {
                echo '빌드 / 배포 Success'
                def Author_ID = sh(script: "git show -s --pretty=%an",
returnStdout: true).trim()
                def Author_Email = sh(script: "git show -s --pretty=%ae",
returnStdout: true).trim()
                mattermostSend(
                    color: 'good',
                    message: "빌드 성공: ${env.JOB_NAME} #${env.BUILD_NUMBER} by
${Author_ID}(${Author_Email})",
                    endpoint: 'mattermost hook endpoint',
                    channel: 'channel'
                )
            }
        }
        failure {
            echo 'Pipeline Execution Failed.'
            script {
                echo '빌드 / 배포 Failed'
```

```
                def Author_ID = sh(script: "git show -s --pretty=%an",
returnStdout: true).trim()
                def Author_Email = sh(script: "git show -s --pretty=%ae",
returnStdout: true).trim()
                mattermostSend(
                    color: 'danger',
                    message: "빌드 실패: ${env.JOB_NAME} #${env.BUILD_NUMBER} by
${Author_ID}(${Author_Email})",
                    endpoint: 'mattermost hook endpoint',
                    channel: 'channel'
                )
            }
        }
    }
}
```

nextjs pipeline

```
pipeline {
    agent any
    environment {
        IP = credentials('server_IP')
    }

    stages {
        // 1단계: Git 클론
        stage('Git Clone') {
            steps {
                git branch: 'dev-next', credentialsId: 'gitlab-credentials', url:
'https://lab.ssafy.com/s11-final/S11P31B105.git'
            }
            post {
                success { echo 'Repository clone 성공!' }
                failure { echo 'Repository clone 실패!' }
            }
        }

        // 2단계: Docker 이미지 빌드
        stage('Build Docker Image') {
            steps {
                dir('./frontend/project105') {
                    sh 'docker build --no-cache -t nextjs .'
                }
                echo 'Docker 이미지 빌드 완료'
            }
            post {
                success { echo '이미지 빌드 성공' }
                failure { echo '이미지 빌드 실패' }
            }
        }
```

```
        // 3단계: 기존 컨테이너 중지 및 제거
        stage('Stop and Remove Existing Container') {
            steps {
                echo '기존 NextJS 컨테이너 중지 및 제거 시작'
                sshagent(credentials: ['ec2']) {
                    sh '''
                        ssh -o StrictHostKeyChecking=no -p 2201 ubuntu@$IP "
                            docker-compose stop nextjs &&
                            docker-compose rm -f nextjs
                        "
                    '''
                }
                echo '기존 NextJS 컨테이너 중지 및 제거 완료'
            }
            post {
                success { echo '컨테이너 중지 및 제거 성공' }
                failure { echo '컨테이너 중지 및 제거 실패' }
            }
        }

        // 4단계: 새로운 컨테이너 배포
        stage('Deploy New Container') {
            steps {
                echo 'NextJS 서비스 배포 시작'
                sshagent(credentials: ['ec2']) {
                    sh '''
                        ssh -o StrictHostKeyChecking=no -p 2201 ubuntu@$IP "
                            docker-compose up -d --remove-orphans nextjs
                        "
                    '''
                }
                echo 'NextJS 서비스 배포 완료'
            }
            post {
                success { echo '배포 성공' }
                failure { echo '배포 실패' }
            }
        }
    }

    post {
        always {
            echo 'Pipeline Execution Complete.'
        }
        success {
            echo 'Pipeline Execution Success.'
            script {
                echo '빌드 / 배포 Success'
                def Author_ID = sh(script: "git show -s --pretty=%an",
returnStdout: true).trim()
                def Author_Email = sh(script: "git show -s --pretty=%ae",
returnStdout: true).trim()
                mattermostSend(
                    color: 'good',
```

```
                    message: "빌드 성공: ${env.JOB_NAME} #${env.BUILD_NUMBER} by
${Author_ID}(${Author_Email})",
                    endpoint: 'mattermost hook endpoint',
                    channel: 'channel'
                )
            }
        }
        failure {
            echo 'Pipeline Execution Failed.'
            script {
                echo '빌드 / 배포 Failed'
                def Author_ID = sh(script: "git show -s --pretty=%an",
returnStdout: true).trim()
                def Author_Email = sh(script: "git show -s --pretty=%ae",
returnStdout: true).trim()
                mattermostSend(
                    color: 'danger',
                    message: "빌드 실패: ${env.JOB_NAME} #${env.BUILD_NUMBER} by
${Author_ID}(${Author_Email})",
                    endpoint: 'mattermost hook endpoint',
                    channel: 'channel'
                )
            }
        }
    }
}
```

spring pipeline

```
pipeline {
    agent any
    environment {
        IP = credentials('server_IP')
    }

    stages {
        // 1단계: Git 클론
        stage('Git Clone') {
            steps {
                git branch: 'dev-be', credentialsId: 'gitlab-credentials', url:
'https://lab.ssafy.com/s11-final/S11P31B105.git'
            }
            post {
                success { echo 'Repository clone 성공!' }
                failure { echo 'Repository clone 실패!' }
            }
        }

        // 2단계: secret.properties 파일 준비
        stage('Prepare secret.properties') {
            steps {
```

```
                script {
                    echo 'secret.properties 파일 준비 시작...'
                    withCredentials([file(credentialsId: 'secret-properties',
variable: 'SECRET_FILE')]) {
                        sh '''
                            chown -R jenkins:jenkins
./backend/spring/wassu/src/main/resources/
                            chmod -R u+w
./backend/spring/wassu/src/main/resources/
                            cp $SECRET_FILE
./backend/spring/wassu/src/main/resources/secret.properties
                        '''
                    }
                }
            }
            post {
                success { echo 'secret.properties 파일 준비 완료!' }
                failure { echo 'secret.properties 파일 준비 실패!' }
            }
        }

        // 3단계: Docker 이미지 빌드
        stage('Build Docker Image') {
            steps {
                dir('./backend/spring/wassu') {
                    sh 'docker build --no-cache -t spring .'
                }
                echo 'Docker 이미지 빌드 완료'
            }
            post {
                success { echo 'Docker 이미지 빌드 성공!' }
                failure { echo 'Docker 이미지 빌드 실패!' }
            }
        }

        // 4단계: 기존 Spring 컨테이너 중지 및 제거
        stage('Stop and Remove Existing Container') {
            steps {
                echo '기존 Spring 컨테이너 중지 및 제거 시작'
                sshagent(credentials: ['ec2']) {
                    sh '''
                        ssh -o StrictHostKeyChecking=no -p 2201 ubuntu@$IP "
                            docker-compose stop spring &&
                            docker-compose rm -f spring
                        "
                    '''
                }
                echo '기존 Spring 컨테이너 중지 및 제거 완료'
            }
            post {
                success { echo '컨테이너 중지 및 제거 성공' }
                failure { echo '컨테이너 중지 및 제거 실패' }
            }
        }
```

```
        // 5단계: 새로운 Spring 컨테이너 배포
        stage('Deploy New Container') {
            steps {
                echo 'Spring 서비스 배포 시작'
                sshagent(credentials: ['ec2']) {
                    sh '''
                        ssh -o StrictHostKeyChecking=no -p 2201 ubuntu@$IP "
                            docker-compose up -d --remove-orphans spring
                        "
                    '''
                }
                echo 'Spring 서비스 배포 완료'
            }
            post {
                success { echo '배포 성공' }
                failure { echo '배포 실패' }
            }
        }
    }

    post {
        always {
            echo 'Pipeline Execution Complete.'
        }
        success {
            echo 'Pipeline Execution Success.'
            script {
                echo '빌드 / 배포 Success'
                def Author_ID = sh(script: "git show -s --pretty=%an",
returnStdout: true).trim()
                def Author_Email = sh(script: "git show -s --pretty=%ae",
returnStdout: true).trim()
                mattermostSend(
                    color: 'good',
                    message: "빌드 성공: ${env.JOB_NAME} #${env.BUILD_NUMBER} by
${Author_ID}(${Author_Email})",
                    endpoint: 'mattermost hook endpoint',
                    channel: 'channel'
                )
            }
        }
        failure {
            echo 'Pipeline Execution Failed.'
            script {
                echo '빌드 / 배포 Failed'
                def Author_ID = sh(script: "git show -s --pretty=%an",
returnStdout: true).trim()
                def Author_Email = sh(script: "git show -s --pretty=%ae",
returnStdout: true).trim()
                mattermostSend(
                    color: 'danger',
                    message: "빌드 실패: ${env.JOB_NAME} #${env.BUILD_NUMBER} by
${Author_ID}(${Author_Email})",
```

```
                    endpoint: 'mattermost hook endpoint',
                    channel: 'channel'
                )
            }
        }
    }
}
```

fastapi pipeline

```
pipeline {
    agent any
    environment {
        IP = credentials('server_IP')
    }

    stages {
        // 1단계: Git 클론
        stage('Git Clone') {
            steps {
                git branch: 'fast_api', credentialsId: 'gitlab-credentials', url:
'https://lab.ssafy.com/s11-final/S11P31B105.git'
            }
            post {
                success { echo 'Repository clone 성공!' }
                failure { echo 'Repository clone 실패!' }
            }
        }

        // 2단계: Docker 이미지 빌드
        stage('Build Docker Image') {
            steps {
                sh 'docker build --no-cache -t fast-api .'
                echo 'Docker 이미지 빌드 완료'
            }
            post {
                success { echo '이미지 빌드 성공' }
                failure { echo '이미지 빌드 실패' }
            }
        }

        // 3단계: 기존 컨테이너 중지 및 제거
        stage('Stop and Remove Existing Container') {
            steps {
                echo '기존 Fast-API 컨테이너 중지 및 제거 시작'
                sshagent(credentials: ['ec2']) {
                    sh '''
                        ssh -o StrictHostKeyChecking=no -p 2201 ubuntu@$IP "
                            docker-compose stop fast-api &&
                            docker-compose rm -f fast-api
                        "
```

```
                    '''
                }
                echo '기존 Fast-API 컨테이너 중지 및 제거 완료'
            }
            post {
                success { echo '컨테이너 중지 및 제거 성공' }
                failure { echo '컨테이너 중지 및 제거 실패' }
            }
        }

        // 4단계: 새로운 컨테이너 배포
        stage('Deploy New Container') {
            steps {
                echo 'Fast-API 서비스 배포 시작'
                sshagent(credentials: ['ec2']) {
                    sh '''
                        ssh -o StrictHostKeyChecking=no -p 2201 ubuntu@$IP "
                            docker-compose up -d --remove-orphans fast-api
                        "
                    '''
                }
                echo 'Fast-API 서비스 배포 완료'
            }
            post {
                success { echo '배포 성공' }
                failure { echo '배포 실패' }
            }
        }
    }

    post {
        always {
            echo 'Pipeline Execution Complete.'
        }
        success {
            echo 'Pipeline Execution Success.'
            script {
                echo '빌드 / 배포 Success'
                def Author_ID = sh(script: "git show -s --pretty=%an",
returnStdout: true).trim()
                def Author_Email = sh(script: "git show -s --pretty=%ae",
returnStdout: true).trim()
                mattermostSend(
                    color: 'good',
                    message: "빌드 성공: ${env.JOB_NAME} #${env.BUILD_NUMBER} by
${Author_ID}(${Author_Email})",
                    endpoint: 'mattermost hook endpoint',
                    channel: 'channel'
                )
            }
        }
        failure {
            echo 'Pipeline Execution Failed.'
            script {
```

```
                echo '빌드 / 배포 Failed'
                def Author_ID = sh(script: "git show -s --pretty=%an",
returnStdout: true).trim()
                def Author_Email = sh(script: "git show -s --pretty=%ae",
returnStdout: true).trim()
                mattermostSend(
                    color: 'danger',
                    message: "빌드 실패: ${env.JOB_NAME} #${env.BUILD_NUMBER} by
${Author_ID}(${Author_Email})",
                    endpoint: 'mattermost hook endpoint',
                    channel: 'channel'
                )
            }
        }
    }
}
```

5. spring secret properties

- gitingore를 통해 업로드 방지
- jenkins credentials로 관리

```
# PostgreSQL
DB_USERNAME=
DB_PASSWORD=
DB_HOST=
DB_PORT=
DB_NAME=

# JWT
JWT_SECRET=
JWT_ACCESS_EXPIRATION=
JWT_REFRESH_EXPIRATION=
JWT_ALGORITHM=

# Mail Post
MAIL_HOST=
MAIL_PASSWORD=

# Amazon S3
S3_BUCKET=
S3_REGION=
S3_ACCESSKEY=
S3_SECRETKEY=

# ElasticSearch
ELASTIC_USER=
ELASTIC_PASSWORD=
ELASTIC_ENDPOINT=
```

```
SERVER_DOMAIN=
REDIS_PORT=
```

6. 외부 서비스

- OPEN AI API