# 포팅 메뉴얼

## 간단 요약 정보

```
# ALBARO 포팅 메뉴얼

## 1. 개요
본 문서는 ALBARO 프로젝트의 배포를 위한 포팅 메뉴얼입니다.
Jenkins를 활용한 CI/CD 파이프라인 구축 및
Docker 기반의 컨테이너화된 서비스 배포 방법을 설명합니다.

## 2. 시스템 요구사항

### 2.1 하드웨어
- CPU: 듀얼코어 이상 권장
- RAM: 8GB 이상 권장
- 저장공간: 최소 20GB 이상의 여유 공간

### 2.2 소프트웨어
- Ubuntu 20.04 LTS 이상
- Docker 20.10.x 이상
- Docker Compose v2.x 이상
- Jenkins 2.x 이상
- Java 11 이상
- Node.js 16.x 이상
- MySQL 8.x
- Redis 최신 버전

## 3. 환경 설정

### 3.1 Docker 설치
sudo apt-get update
sudo apt-get install docker.io
sudo systemctl start docker
sudo systemctl enable docker

### 3.2 Docker Compose 설치
sudo curl -L "https://github.com/docker/compose/releases/download/v2.5.0/docker-compose-$(uname -s)-$(uname -m)
sudo chmod +x /usr/local/bin/docker-compose

### 3.3 Jenkins 설치 및 설정
1. Jenkins 컨테이너 실행:
docker-compose up -d jenkins

2. Jenkins 초기 비밀번호 확인:
docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword

3. Jenkins 플러그인 설치:
  - Docker Pipeline
  - Git
  - Pipeline
  - Credentials Plugin

### 3.4 SSL 인증서 설정
1. Let's Encrypt 인증서 발급:
sudo certbot certonly --standalone -d [도메인명]
```

2. 인증서 위치 확인:
/etc/letsencrypt/live/[도메인명]/

## 4. 프로젝트 구성 및 배포

### 4.1 프로젝트 구조
```
project/
├── backend/         # Spring Boot 애플리케이션
├── frontend/        # Next.js 애플리케이션
├── jenkins/         # Jenkins 설정 파일
├── nginx/           # Nginx 설정 파일
└── face_recognition/ # 얼굴 인식 서비스
```

### 4.2 환경 변수 설정
1. Jenkins Credentials에 다음 항목 추가:
   - NEXT_PUBLIC_KAKAO_KEY
   - Gitlab 접근 credentials

2. .env 파일 설정:
   - NEXT_PUBLIC_KAKAO_KEY
   - NEXT_PUBLIC_API_URL

### 4.3 배포 프로세스
1. Jenkins 파이프라인 생성:
   - 새로운 Pipeline 항목 생성
   - GitLab Repository URL 설정
   - Jenkinsfile 경로 지정

2. 배포 순서:
   - 소스코드 체크아웃
   - Backend 빌드
   - 환경 변수 설정
   - Docker 컨테이너 실행
   - Nginx 설정 적용

## 5. 서비스 구성요소

### 5.1 서비스 포트
- Jenkins: 8081
- Frontend: 3000
- Backend: 8080
- MySQL: 3306
- Redis: 6379
- Face Recognition: 5000
- Nginx: 80, 443

### 5.2 네트워크 구성
- Docker network: app-network (bridge mode)

## 6. 문제해결 및 디버깅

### 6.1 로그 확인
```
# 전체 컨테이너 상태 확인
docker ps

# 특정 서비스 로그 확인
docker-compose logs [서비스명]
```

```
# Jenkins 빌드 로그 확인
Jenkins 웹 인터페이스 > 빌드 히스토리 > Console Output

### 6.2 일반적인 문제해결
1. 컨테이너 시작 실패
   - 로그 확인
   - 포트 충돌 확인
   - 환경 변수 설정 확인

2. 빌드 실패
   - Gradle 빌드 로그 확인
   - 의존성 문제 확인
   - 권한 설정 확인

## 7. 유지보수

### 7.1 백업
- Jenkins 설정: /var/jenkins_home
- MySQL 데이터: /var/lib/mysql
- SSL 인증서: /etc/letsencrypt

### 7.2 모니터링
- 컨테이너 상태 모니터링
- 시스템 리소스 사용량 확인
- 로그 모니터링

## 8. 참고사항
- SSL 인증서는 3개월마다 자동 갱신됨
- Docker 이미지는 주기적으로 최신 버전으로 업데이트 필요
- 보안 업데이트 및 패치 적용 필요

## 9. 문의 및 지원
- 기술 지원: 010-9243-9104 AIBaro팀 인프라 담당
- 문서 최종 수정일: 2025.02.21
```

---

## 1. Jenkins 설정 상세

### 1.1 Jenkins 컨테이너 구성

```
# docker-compose.yml
services:
  jenkins:
    build: ./jenkins
    user: jenkins
    ports:
      - "8081:8080"
    volumes:
      - jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
      - /etc/letsencrypt:/etc/letsencrypt:ro
    environment:
      - JAVA_OPTS=-Djava.awt.headless=true
    privileged: true
```

### 1.2 Jenkins Dockerfile 설정

```
# jenkins/Dockerfile
FROM jenkins/jenkins:lts
USER root

# Docker 설치
RUN apt-get update && \
    apt-get install -y apt-transport-https ca-certificates curl software-properties-common && \
    curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add - && \
    add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian $(lsb_release -cs) stable" && \
    apt-get update && \
    apt-get install -y docker-ce-cli

# Docker 그룹에 Jenkins 사용자 추가
RUN usermod -aG docker jenkins
```

---

## 2. Backend 서비스 구성

### 2.1 Backend Dockerfile

```
FROM openjdk:17-jdk-slim
WORKDIR /app

# wait-for-it.sh 스크립트 설정
COPY wait-for-it.sh /usr/local/bin/wait-for-it.sh
RUN chmod +x /usr/local/bin/wait-for-it.sh

# 애플리케이션 빌드 및 실행
COPY build/libs/backend-0.0.1-SNAPSHOT.jar /app/app.jar
ENTRYPOINT ["/usr/local/bin/wait-for-it.sh", "db:3306", "--", "java", "-jar", "/app/app.jar"]
```

### 2.2 Backend 환경 설정

```
# application.properties
spring.datasource.url=jdbc:mysql://db:3306/albaro
spring.datasource.username=비밀
spring.datasource.password=비밀
spring.jpa.hibernate.ddl-auto=update
```

---

## 3. Face Recognition 서비스

### 3.1 Face Recognition Dockerfile

```
FROM python:3.9-slim
WORKDIR /app

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .
```

```
EXPOSE 5000

CMD ["python", "app.py"]
```

### 3.2 Face Recognition 의존성

```
flask
torch==2.2.0
torchvision==0.17.0
facenet-pytorch
pillow
flask-cors
```

---

## 4. Frontend 구성

### 4.1 Frontend Dockerfile

```
FROM node:20-alpine
WORKDIR /app

# 의존성 설치
COPY package*.json ./
RUN npm install

# 소스 코드 복사 및 빌드
COPY . .
RUN npm run build

EXPOSE 3000
CMD ["npm", "start"]
```

### 4.2 Frontend 환경 변수

```
NEXT_PUBLIC_API_URL= 비밀
NEXT_PUBLIC_WS_URL= 비밀
NEXT_PUBLIC_KAKAO_KEY= 비밀
```

---

## 5. Nginx 설정

### 5.1 Nginx 설정 파일

```
# nginx/conf.d/default.conf
map $http_upgrade $connection_upgrade {
    default upgrade;
    ''      close;
}

upstream backend-ws {
    server backend:8080;
    keepalive 32;
}
```

```
server {
    listen 443 ssl;
    server_name your-domain.com;

    ssl_certificate /etc/letsencrypt/live/your-domain.com/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/your-domain.com/privkey.pem;

    # WebSocket 설정
    location /ws {
        proxy_pass http://backend-ws;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection $connection_upgrade;
    }

    # API 프록시
    location /api {
        proxy_pass http://backend:8080;
    }

    # Frontend 프록시
    location / {
        proxy_pass http://frontend:3000;
    }
}
```

---

## 6. 전체 서비스 구성

### 6.1 docker-compose.yml 전체 구성

```
version: '3.8'

services:
  jenkins:
    build: ./jenkins
    # Jenkins 설정 (위 참조)

  backend:
    build: ./backend
    depends_on:
      - db
      - redis
    environment:
      - SPRING_PROFILES_ACTIVE=prod
      - MYSQL_USER=${MYSQL_USER}
      - MYSQL_PASSWORD=${MYSQL_PASSWORD}

  frontend:
    build: ./frontend
    environment:
      - NODE_ENV=production
      - NEXT_PUBLIC_API_URL=${NEXT_PUBLIC_API_URL}

  face_recognition:
    build: ./face_recognition
    ports:
```

```
    - "5000:5000"

nginx:
  build: ./nginx
  ports:
    - "80:80"
    - "443:443"
  volumes:
    - /etc/letsencrypt:/etc/letsencrypt:ro
  depends_on:
    - backend
    - frontend

db:
  image: mysql:8
  environment:
    - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}
    - MYSQL_DATABASE=albaro
  volumes:
    - mysql_data:/var/lib/mysql

  redis:
    image: redis:latest
    volumes:
      - redis_data:/data

volumes:
  jenkins_home:
  mysql_data:
  redis_data:
```

---

## 7. 배포 프로세스

### 7.1 Jenkins Pipeline 스크립트

```
pipeline {
    agent any

    environment {
    EC2_IP = 'i12b105.p.ssafy.io'
    SSH_KEY = 'C:/Users/SSAFY/Desktop/I12B105T.pem'
    PROJECT_PATH = '/var/jenkins_home/workspace/jenkinsTest/jenkins'  // Jenkins 컨테이너 내부 절대 경로
    GIT_REPO_URL = 'https://lab.ssafy.com/s12-webmobile1-sub1/S12P11B105.git'
    GIT_BRANCH = 'develop'
    DOCKER_COMPOSE_PATH = '/usr/local/bin/docker-compose'
}

stages {
    stage('Checkout') {
        steps {
            git url: "${GIT_REPO_URL}", branch: "${GIT_BRANCH}", credentialsId: 'Gitlab'
        }
    }


    stage('Build') {
```

```
        steps {
            script {
                // backend 디렉토리로 이동하여 gradlew 실행
                dir('backend') {
                    // gradlew 파일에 실행 권한 부여
                    sh 'chmod +x gradlew'

                    // Gradle 빌드 수행
                    sh './gradlew build -x test --no-daemon'

                     // JAR 파일 존재 여부 확인
                    sh 'ls -l build/libs || echo "JAR 파일이 존재하지 않습니다!"'
                }
            }
        }
    }

    stage('.env 파일 생성') {
        steps {
            withCredentials([string(credentialsId: 'NEXT_PUBLIC_KAKAO_KEY', variable: 'NEXT_PUBLIC_KAKAO_KEY')]) {
                script {
                    sh '''
                    echo "NEXT_PUBLIC_KAKAO_KEY=$NEXT_PUBLIC_KAKAO_KEY" > /var/jenkins_home/workspace/jenkinsT
                    cp /var/jenkins_home/workspace/jenkinsTest/jenkins/.env /var/jenkins_home/workspace/jenkinsTest/fronte
                    echo "NEXT_PUBLIC_API_URL=https://i12b105.p.ssafy.io" >> /var/jenkins_home/workspace/jenkinsTest/fro
                    '''
                }
            }
        }
    }

    stage('기존 컨테이너 중지 및 제거') {
        steps {
            script {
                sh '''
                cd /var/jenkins_home/workspace/jenkinsTest/jenkins
                $DOCKER_COMPOSE_PATH down
                '''
            }
        }
    }

    stage('컨테이너 빌드 및 시작') {
        steps {
            script {
                sh '''
                cd /var/jenkins_home/workspace/jenkinsTest/jenkins
                $DOCKER_COMPOSE_PATH up --build -d
                '''
            }
        }
    }

    stage('Deploy Nginx') {
        steps {
            script {
                sh '''
                cd /var/jenkins_home/workspace/jenkinsTest/jenkins
```

```
                $DOCKER_COMPOSE_PATH up -d nginx
                '''
            }
        }
    }


    stage('Deploy') {
        steps {
            // Docker 네트워크 생성 확인
            sh 'docker network ls'
            // 컨테이너 상태 확인
            sh 'docker ps'
        }
    }

}

    post {
        success {
            echo '빌드 및 배포가 성공적으로 완료되었습니다! :)'
        }
        failure {
            echo '빌드 또는 배포에 실패했습니다!  ㅡ.ㅡ'
        }
    }
}
```

### 7.2 배포 후 확인사항

- 서비스 상태 확인

```
docker-compose ps
```

- 로그 모니터링

```
docker-compose logs -f [service-name]
```