

# Yaldi 포팅 매뉴얼 작성

## 목차

### 1. 개요

#### 1-1. 프로젝트 개요

#### 1-2. 사용 기술

#### 1-3. 사전 준비사항

### 2. 환경 설정 및 빌드

#### 2-1. 프로젝트 구조

#### 2-2. 배포 환경 구축 순서

## 1. 개요

### 1-1. 프로젝트 개요

Yaldi는 AI 에이전트 기반 실시간 ERD 협업툴입니다.

- 서비스 URL : <https://yaldi.kr>
- Jenkins admin URL: <https://jenkins.yaldi.kr>

### 1-2. 사용 기술

인프라 서비스

- Nginx (리버스 프록시, SSL 종료)

- Jenkins (CI/CD)

### 애플리케이션 서비스

- Backend: Spring Boot 3.x + Java 21
- Frontend: React + Vite
- AI Service: FastAPI + Python 3.11

### 데이터 스토어

- PostgreSQL 16 (+ pgvector)
- Redis 7
- Kafka 3.8.1 (KRaft 모드)
- Elasticsearch 8.11
- Neo4j 5.15 (AI Graph RAG용)

### 모니터링

- Kafka UI

## 1-3. 사전 준비 사항

### 1-3-1. 도메인 및 DNS 설정

다음 서브도메인을 DNS에 등록해야 합니다.

yaldi.kr → A Record → [서버 IP]  
www.yaldi.kr → A Record → [서버 IP]  
api.yaldi.kr → A Record → [서버 IP]  
ai.yaldi.kr → A Record → [서버 IP]  
jenkins.yaldi.kr → A Record → [서버 IP]

### 1-3-2. 필요한 외부 서비스 계정

- **GitLab 계정**
  - 프로젝트 저장소 접근 권한

- Personal Access Token 발급
- **OAuth 제공자**
  - Google Cloud Console (OAuth 2.0 클라이언트 ID)
  - GitHub OAuth Apps
- **AWS S3 버킷**
  - 파일 업로드용
  - IAM Access Key/Secret Key
  - presigned 다운로드 관련 CORS 처리 필요

### 1-3-3. 환경 변수 준비

다음 정보들을 미리 준비해야 합니다.

```
# Database
DB_NAME=yaldi
DB_USER=postgres
DB_PASSWORD=<strong_password>

# Redis
REDIS_PASSWORD=<strong_password>

# JWT
JWT_SECRET=<256bit_random_string>

# Google OAuth
GOOGLE_CLIENT_ID=<google_client_id>
GOOGLE_CLIENT_SECRET=<google_client_secret>
GOOGLE_REDIRECT_URI=https://api.yaldi.kr/login/oauth2/code/google

# GitHub OAuth
GITHUB_CLIENT_ID=<github_client_id>
GITHUB_CLIENT_SECRET=<github_client_secret>
GITHUB_REDIRECT_URI=https://api.yaldi.kr/login/oauth2/code/github
```

```

# AWS S3
AWS_ACCESS_KEY=<aws_access_key>
AWS_SECRET_KEY=<aws_secret_key>
AWS_S3_BUCKET_NAME=<bucket_name>

# AI Service
GMS_API_KEY=<ssafy_gms_api_key>
FASTAPI_BACKEND_API_KEY=<random_api_key>

# CORS
CORS_ALLOWED_ORIGINS=https://yaldi.kr,https://api.yaldi.kr
OAUTH2_REDIRECT_URI=https://yaldi.kr/oauth2/redirect

# Kafka
KAFKA_CLUSTER_ID=<random_22char_base64>

```

## 2. 환경설정 및 빌드

### 2-1. 프로젝트 구조

- GitLab Front-End 폴더 내 배포 관련 파일 없음 (정적 서빙만 담당)
- GitLab Back-End 폴더 (Spring Boot)

```

Back-End/
├── yaldi/
│   ├── Dockerfile #(Java 21 + Spring Boot)
│   ├── .dockerignore
│   ├── docker-compose.yml # 백엔드 로컬 빌드테스트용 도커컴포즈(배포용 X)
│   ├── build.gradle
│   ├── settings.gradle
│   └── src/

```

```

├── main/
│   ├── resources/
│   │   ├── application.yml
│   │   └── application-prod.yml
│   └── test/

```

- GitLab AI 폴더 (Python fastAPI)

```

AI/
├── yaldi/
│   ├── Dockerfile #(Python 3.11 + uvicorn)
│   ├── .dockerignore
│   ├── docker-compose.yml # AI 로컬 빌드테스트용 도커컴포즈(배포용 X)
│   ├── requirements.txt
│   ├── main.py #(FastAPI 진입점)
│   ├── agents/ #(AI 에이전트)
│   ├── api/v1/ #(API 엔드포인트)
│   ├── config/ #(설정)
│   ├── core/llm/ #(LLM 코어)
│   ├── models/ #(요청/응답 모델)
│   ├── prompts/ #(프롬프트)
│   ├── rag/ #(RAG 시스템)
│   ├── services/ #(비즈니스 로직)
│   └── utils/ #(유틸리티)

```

- EC2 내부

```

/home/ubuntu/yaldi
├── nginx/
│   ├── conf.d/
│   │   └── all-domains.conf # nginx 설정파일
│   └── html
├── volumes/ # bind mount volumes
│   ├── backend-logs
│   └── elasticsearch

```

```

|   |— kafka
|   |— mysql-init
|   |— nginx-logs
|   |— zookeeper
|   |— certbot-www
|   |— fastapi-logs
|   |— jenkins
|   |— letsencrypt
|   |— neo4j
|   |— postgres
|   |— redis
|— frontend/          # 프론트엔드 정적파일 마운트
|— monitoring/
|— jenkins/          # Android (모바일 티켓 구현)
|   |— Dockerfile    # Jenkins 도커파일
|— scripts/          # Android (모바일 티켓 구현)
|— docker-compose.yml # 컨테이너 오케스트레이션

```

## 2-2. 배포 환경 구축 순서

### EC2 초기 셋업

#### ※ 방화벽 설정

```

# 방화벽 설정 (필요한 포트만 열기)
sudo ufw --force enable
sudo ufw allow 22/tcp  # SSH
sudo ufw allow 80/tcp  # HTTP
sudo ufw allow 443/tcp # HTTPS
# sudo ufw allow 6443/tcp # k3s API server
# 내부 포트들 (나중에 리버스 프록시로 통합)
# Spring Boot: 8080 (표준 유지)
sudo ufw allow 9090/tcp # Jenkins: 9090 (양보)

```

## 1. 기본 시스템 업데이트

```
sudo apt update && sudo apt upgrade -y
```

## 2. 기본 도구들 설치 (git, vim, htop)

```
sudo apt install -y git vim htop
```

## 3. Docker 설치 및 Docker 사용자 권한 설정

```
# Docker 공식 GPG 키 추가
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```

```
# Docker 저장소 추가
echo "deb [arch=amd64 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
```

```
# Docker 설치
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io
```

```
# Docker 서비스 시작 및 활성화
sudo systemctl start docker
sudo systemctl enable docker
```

## 4. Docker Compose 설치 (standalone 방식) + docker compose 둘다 설치

```
# Docker Compose 설치 (최신 버전)
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-co
```

```
compose
```

```
# 실행 권한 부여
```

```
sudo chmod +x /usr/local/bin/docker-compose
```

```
# 1. V2 플러그인 설치
```

```
sudo apt update
```

```
sudo apt install -y docker-compose-plugin
```

```
# 2. 확인
```

```
docker compose version
```

```
# Docker Compose version v2.29.x
```

```
# 3. 테스트
```

```
docker compose --help
```

```
# 4. (선택) 기존 standalone 제거
```

```
# sudo rm /usr/local/bin/docker-compose
```

```
# → 안 지워도 됨! 둘 다 있어도 OK
```

## 5. 사용자 Docker 권한설정

```
# Docker 사용자 권한 설정
```

```
# 현재 사용자를 docker 그룹에 추가 (sudo 없이 docker 명령어 사용)
```

```
sudo usermod -aG docker $USER
```

```
# 이후 로그아웃 -> 재로그인 진행 필요. SSH 세션 끊고 다시 접속 하기
```

## EC2 내 Jenkins 초기 설정

```
mkdir home/ubuntu/yaldi
```

```
cd yaldi
```

```
# exec 폴더 내 docker-compose.yml 넣기
```

```
vi docker-compose.yml
```

```
# 데이터 저장 디렉토리 생성
```

```
mkdir -p volumes/{jenkins,postgres,redis,backend-logs,nginx-logs,letsencrypt,certbot-www,kafka,mysql-init,zookeeper,fastapi-logs,elasticsearch}
```

```
# Neo4j 데이터 저장 디렉토리
```

```
sudo mkdir -p /home/ubuntu/yaldi/volumes/neo4j/data
```

```
sudo mkdir -p /home/ubuntu/yaldi/volumes/neo4j/logs
```

```
mkdir -p nginx/conf.d
```

```
mkdir -p scripts
```

```
mkdir -p frontend/dist
```

```
chmod -R 700 volumes/
```

## ✓ GitLab Token 발급

1. 프로젝트 내 메뉴 > Settings > Access tokens
2. 우측 상단 'Add new token' 선택
3. Token 이름, 설명, 만료기간, role(developer), read\_repository 설정
4. create project access token 클릭 후 토큰 안전한 곳에 복사해두기

→ Jenkins 파이프라인에 등록!

## ✓ 데이터 디렉토리 권한 부여

```
# postgresql
```

```
$ sudo chown -R 999:999 /home/ubuntu/yaldi/volumes/postgres
```

```
# 엘라스틱서치
```

```
$ sudo chown -R 1000:1000 /home/ubuntu/yaldi/volumes/elasticsearch
```

```
$ 카프카/주키퍼
```

```
$ sudo chown -R 1000:1000 /home/ubuntu/yaldi/volumes/zookeeper
```

```
$ sudo chown -R 1000:1000 /home/ubuntu/yaldi/volumes/kafka

# FastAPI
$ sudo chown -R 1000:1000 /home/ubuntu/yaldi/volumes/fastapi-logs

# Neo4j
$ sudo chown -R 7474:7474 /home/ubuntu/yaldi/volumes/neo4j
```

등등등...

## ✓ Jenkins만 먼저 띄우기

→ 접속 경로: <http://yaldi.kr:9090>

```
# jenkins 도커파일 만들어두기 (exec 폴더 내 dockerfile 넣기)
cd ~/yaldi/volumes/jenkins
vi Dockerfile

# jenkins 띄우기
$ docker compose --profile jenkins up -d jenkins

# 초기 비밀번호 확인
$ docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

→ 플러그인 설치 및 파이프라인 생성 진행, 민감한 환경변수 Jenkins Credentials 처리

S	W	Name ↓	최근 성공	최근 실패	최근 소요 시간	
✓	☀	yaldi-ai	28 min #56	4 days 1 hr #10	16 sec	▶
✓	☀	yaldi-backend	21 min #104	9 days 0 hr #36	41 sec	▶
✓	☀	yaldi-frontend	1 hr 16 min #72	3 days 0 hr #50	34 sec	▶

## ✓ jenkins 미러사이트 오류 처리

: plugins 설치가 자꾸 제대로 안됨 → 중국 미러사이트 오류

### 1단계: Jenkins 컨테이너 접속

```
docker exec -it jenkins bash
```

### 2단계: CA 인증서 다운로드 및 설정 + Update Center 미러 설정

```
# Jenkins 컨테이너 안에서
cd /var/jenkins_home

# 1. CA 디렉토리 생성
mkdir -p update-center-rootCAs

# 2. curl로 다운로드 (wget 대신)
curl -L https://cdn.jsdelivr.net/gh/lework/jenkins-update-center/rootCA/update-center.crt \
-o update-center-rootCAs/update-center.crt

# 3. Update Center 설정
cat > hudson.model.UpdateCenter.xml << 'EOF'
<?xml version='1.1' encoding='UTF-8'?>
<sites>
  <site>
    <id>default</id>
    <url>https://raw.githubusercontent.com/lework/jenkins-update-center/master/updates/tencent/update-center.json</url>
  </site>
</sites>
EOF

# 4. 권한 설정
chown -R jenkins:jenkins update-center-rootCAs/
```

```
chown jenkins:jenkins hudson.model.UpdateCenter.xml
```

# 5. 캐시 삭제

```
rm -rf updates/
```

# 6. 확인

```
pwd # /var/jenkins_home 인지 확인
```

```
ls -la update-center-rootCAs/
```

```
cat hudson.model.UpdateCenter.xml
```

```
exit
```

### 3단계: Jenkins 재시작

```
docker compose --profile jenkins restart
```

## TLS 발급

※ DNS 설정 및 전파 확인 후 진행!

### 1단계: Nginx 설정 준비(certbot 발급용)

```
cat > /home/ubuntu/yaldi/nginx/conf.d/all-domains.conf << 'EOF'
```

```
# 메인 + www
```

```
server {
```

```
    listen 80;
```

```
    server_name yaldi.kr www.yaldi.kr;
```

```
    location /.well-known/acme-challenge/ {
```

```
        root /var/www/certbot;
```

```
    }
```

```
    location / {
```

```
        return 301 https://yaldi.kr$request_uri;
```

```
    }
```

```

}

# Jenkins
server {
    listen 80;
    server_name jenkins.yaldi.kr;

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

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

# API
server {
    listen 80;
    server_name api.yaldi.kr;

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

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

# 설정 테스트
docker exec nginx nginx -t

# Nginx 재시작
docker restart nginx

```

## 권한 임시 변경 후 인증서 발급

# 1. 권한 임시 755로 변경 (certbot이 쓸 수 있게)

```
sudo chmod 755 /home/ubuntu/yaldi/volumes/certbot-www
```

# 2. Dry-run으로 먼저 테스트 (안전!)

```
docker run --rm \
-v /home/ubuntu/yaldi/volumes/letsencrypt:/etc/letsencrypt \
-v /home/ubuntu/yaldi/volumes/certbot-www:/var/www/certbot \
certbot/certbot certonly --webroot \
-w /var/www/certbot \
-d yaldi.kr \
-d www.yaldi.kr \
-d jenkins.yaldi.kr \
-d api.yaldi.kr \
-d ai.yaldi.kr \
--dry-run \
--expand \
--agree-tos \
--email [본인이메일] \
--non-interactive
```

# 3. Dry-run 성공하면 실제 발급

```
docker run --rm \
-v /home/ubuntu/yaldi/volumes/letsencrypt:/etc/letsencrypt \
-v /home/ubuntu/yaldi/volumes/certbot-www:/var/www/certbot \
certbot/certbot certonly --webroot \
-w /var/www/certbot \
-d yaldi.kr \
-d www.yaldi.kr \
-d jenkins.yaldi.kr \
-d api.yaldi.kr \
-d ai.yaldi.kr \
--expand \
--force-renewal \
--agree-tos \
--email [본인이메일] \
--non-interactive
```

# 4. 권한 다시 750으로 복구

```
sudo chmod 750 /home/ubuntu/yaldi/volumes/certbot-www
```

# 5. 인증서 확인

```
sudo ls -la /home/ubuntu/yaldi/volumes/letsencrypt/live/yaldi.kr/
```

```
sudo cat /home/ubuntu/yaldi/volumes/letsencrypt/live/yaldi.kr/cert.pem | openssl x509 -noout -text | grep DNS
```

이후 nginx 설정에 https 추가

## ✅ Nginx, DB 등은 EC2에서 직접 띄우기

- spring boot, fastAPI는 환경변수처리때문에 파이프라인에서 띄워야 함!

# 예시

```
docker compose up -d postgres
```

### ※ 초기 DB 관련 처리

#### 1. EC2 내 .env 설정

# .env는 계속 있어야 함 (postgres/redis가 계속 읽음)

```
cat > /home/ubuntu/yaldi/.env << 'EOF'
```

```
DB_NAME=yaldi
```

```
DB_USER=postgres
```

```
DB_PASSWORD=your_secure_password_here
```

```
EOF
```

```
chmod 600 /home/ubuntu/yaldi/.env
```

# ⚠️ 삭제하면 안 됨! 재부팅 후 db 시작 시 필요

#### 2. systemd 서비스 설정

```
sudo tee /etc/systemd/system/yaldi-db.service << 'EOF'
[Unit]
Description=Yaldi Database Services (PostgreSQL & Redis)
After=docker.service
Requires=docker.service

[Service]
Type=oneshot
RemainAfterExit=yes
WorkingDirectory=/home/ubuntu/yaldi
ExecStart=/usr/bin/docker compose --profile db up -d
ExecStop=/usr/bin/docker compose --profile db down
TimeoutStartSec=0

[Install]
WantedBy=multi-user.target
EOF

sudo systemctl daemon-reload
sudo systemctl enable yaldi-db
sudo systemctl start yaldi-db
sudo systemctl status yaldi-db
```