**🚀 AWS + Docker 환경 구축 가이드**

**📋 목차**

1. [AWS EC2 인스턴스 생성](https://claude.ai/chat/805c9e81-fc26-4d82-9285-3ddab39995f6#aws-ec2-%EC%9D%B8%EC%8A%A4%ED%84%B4%EC%8A%A4-%EC%83%9D%EC%84%B1)
2. [Docker 환경 설정](https://claude.ai/chat/805c9e81-fc26-4d82-9285-3ddab39995f6#docker-%ED%99%98%EA%B2%BD-%EC%84%A4%EC%A0%95)
3. [소셜 로그인 설정](https://claude.ai/chat/805c9e81-fc26-4d82-9285-3ddab39995f6#%EC%86%8C%EC%85%9C-%EB%A1%9C%EA%B7%B8%EC%9D%B8-%EC%84%A4%EC%A0%95)
4. [프로젝트 Dockerization](https://claude.ai/chat/805c9e81-fc26-4d82-9285-3ddab39995f6#%ED%94%84%EB%A1%9C%EC%A0%9D%ED%8A%B8-dockerization)
5. [CI/CD 파이프라인](https://claude.ai/chat/805c9e81-fc26-4d82-9285-3ddab39995f6#cicd-%ED%8C%8C%EC%9D%B4%ED%94%84%EB%9D%BC%EC%9D%B8)

**🔧 AWS EC2 인스턴스 생성**

**1단계: EC2 인스턴스 설정**

# 인스턴스 사양 (추천)

- 인스턴스 타입: t3.medium (2 vCPU, 4GB RAM)

- 운영체제: Ubuntu 22.04 LTS

- 스토리지: 30GB gp3

- 보안 그룹: HTTP(80), HTTPS(443), SSH(22), Custom(8080)

**2단계: 보안 그룹 설정**

{

"SecurityGroupRules": [

{

"IpProtocol": "tcp",

"FromPort": 22,

"ToPort": 22,

"CidrIp": "0.0.0.0/0",

"Description": "SSH"

},

{

"IpProtocol": "tcp",

"FromPort": 80,

"ToPort": 80,

"CidrIp": "0.0.0.0/0",

"Description": "HTTP"

},

{

"IpProtocol": "tcp",

"FromPort": 443,

"ToPort": 443,

"CidrIp": "0.0.0.0/0",

"Description": "HTTPS"

},

{

"IpProtocol": "tcp",

"FromPort": 8080,

"ToPort": 8080,

"CidrIp": "0.0.0.0/0",

"Description": "Spring Boot App"

}

]

}

**3단계: Elastic IP 할당**

# AWS CLI로 Elastic IP 할당

aws ec2 allocate-address --domain vpc

aws ec2 associate-address --instance-id i-1234567890abcdef0 --allocation-id eipalloc-12345678

**🐳 Docker 환경 설정**

**1단계: 서버 접속 및 기본 설정**

# EC2 인스턴스 접속

ssh -i "your-key.pem" ubuntu@your-ec2-public-ip

# 시스템 업데이트

sudo apt update && sudo apt upgrade -y

# 필수 패키지 설치

sudo apt install -y curl wget git vim

**2단계: Docker 설치**

# 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 Compose 설치

sudo curl -L "https://github.com/docker/compose/releases/download/v2.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

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

# 사용자를 docker 그룹에 추가

sudo usermod -aG docker ubuntu

newgrp docker

# Docker 서비스 시작 및 부팅시 자동 시작 설정

sudo systemctl start docker

sudo systemctl enable docker

**3단계: 환경 변수 설정**

# 환경 변수 파일 생성

sudo vim /etc/environment

# 다음 내용 추가

SPRING\_PROFILES\_ACTIVE=prod

DB\_HOST=mysql-container

DB\_USERNAME=portfolio

DB\_PASSWORD=your-secure-password

REDIS\_HOST=redis-container

JWT\_SECRET=your-jwt-secret-key

**🔐 소셜 로그인 설정**

**1. Google OAuth 2.0 설정**

**Google Cloud Console 설정**

1. [Google Cloud Console](https://console.cloud.google.com/) 접속
2. 새 프로젝트 생성 또는 기존 프로젝트 선택
3. "API 및 서비스" > "사용 설정된 API" > "Google+ API" 활성화
4. "사용자 인증 정보" > "사용자 인증 정보 만들기" > "OAuth 클라이언트 ID"

**OAuth 클라이언트 설정**

# application-prod.yml

spring:

security:

oauth2:

client:

registration:

google:

client-id: ${GOOGLE\_CLIENT\_ID}

client-secret: ${GOOGLE\_CLIENT\_SECRET}

scope: profile, email

redirect-uri: "https://your-domain.com/login/oauth2/code/google"

**2. 카카오 OAuth 설정**

**카카오 개발자 센터 설정**

1. [카카오 개발자 센터](https://developers.kakao.com/) 접속
2. 애플리케이션 추가
3. "제품 설정" > "카카오 로그인" 활성화
4. Redirect URI 설정: https://your-domain.com/login/oauth2/code/kakao

# application-prod.yml 추가

kakao:

client-id: ${KAKAO\_CLIENT\_ID}

client-secret: ${KAKAO\_CLIENT\_SECRET}

authorization-grant-type: authorization\_code

redirect-uri: "https://your-domain.com/login/oauth2/code/kakao"

scope: profile\_nickname, account\_email

provider:

kakao:

authorization-uri: https://kauth.kakao.com/oauth/authorize

token-uri: https://kauth.kakao.com/oauth/token

user-info-uri: https://kapi.kakao.com/v2/user/me

user-name-attribute: id

**3. 네이버 OAuth 설정**

**네이버 개발자 센터 설정**

1. [네이버 개발자 센터](https://developers.naver.com/) 접속
2. "애플리케이션 등록"
3. "서비스 URL" 설정
4. "Callback URL" 설정: https://your-domain.com/login/oauth2/code/naver

# application-prod.yml 추가

naver:

client-id: ${NAVER\_CLIENT\_ID}

client-secret: ${NAVER\_CLIENT\_SECRET}

authorization-grant-type: authorization\_code

redirect-uri: "https://your-domain.com/login/oauth2/code/naver"

scope: name, email

provider:

naver:

authorization-uri: https://nid.naver.com/oauth2.0/authorize

token-uri: https://nid.naver.com/oauth2.0/token

user-info-uri: https://openapi.naver.com/v1/nid/me

user-name-attribute: response

**📦 프로젝트 Dockerization**

**1. Dockerfile 작성**

# Dockerfile

FROM openjdk:17-jdk-slim

# 작업 디렉토리 설정

WORKDIR /app

# 필요한 패키지 설치

RUN apt-get update && apt-get install -y curl && rm -rf /var/lib/apt/lists/\*

# Gradle Wrapper와 소스코드 복사

COPY gradlew .

COPY gradle gradle

COPY build.gradle .

COPY settings.gradle .

COPY src src

# Gradle Wrapper 실행 권한 부여

RUN chmod +x ./gradlew

# 애플리케이션 빌드

RUN ./gradlew bootJar --no-daemon

# JAR 파일을 app.jar로 복사

RUN cp build/libs/\*.jar app.jar

# 포트 노출

EXPOSE 8080

# 애플리케이션 실행

ENTRYPOINT ["java", "-jar", "app.jar"]

**2. docker-compose.yml 작성**

version: '3.8'

services:

# MySQL 데이터베이스

mysql:

image: mysql:8.0

container\_name: mysql-container

environment:

MYSQL\_ROOT\_PASSWORD: rootpassword

MYSQL\_DATABASE: portfolio\_db

MYSQL\_USER: portfolio

MYSQL\_PASSWORD: portfolio123!

ports:

- "3306:3306"

volumes:

- mysql\_data:/var/lib/mysql

- ./init.sql:/docker-entrypoint-initdb.d/init.sql

restart: unless-stopped

command: --default-authentication-plugin=mysql\_native\_password

# Redis 캐시

redis:

image: redis:7-alpine

container\_name: redis-container

ports:

- "6379:6379"

volumes:

- redis\_data:/data

restart: unless-stopped

# Spring Boot 애플리케이션

app:

build: .

container\_name: portfolio-app

depends\_on:

- mysql

- redis

ports:

- "8080:8080"

environment:

SPRING\_PROFILES\_ACTIVE: prod

DB\_HOST: mysql

DB\_PORT: 3306

DB\_NAME: portfolio\_db

DB\_USERNAME: portfolio

DB\_PASSWORD: portfolio123!

REDIS\_HOST: redis

REDIS\_PORT: 6379

JWT\_SECRET: your-super-secret-jwt-key-here

GOOGLE\_CLIENT\_ID: ${GOOGLE\_CLIENT\_ID}

GOOGLE\_CLIENT\_SECRET: ${GOOGLE\_CLIENT\_SECRET}

KAKAO\_CLIENT\_ID: ${KAKAO\_CLIENT\_ID}

KAKAO\_CLIENT\_SECRET: ${KAKAO\_CLIENT\_SECRET}

NAVER\_CLIENT\_ID: ${NAVER\_CLIENT\_ID}

NAVER\_CLIENT\_SECRET: ${NAVER\_CLIENT\_SECRET}

restart: unless-stopped

volumes:

- ./logs:/app/logs

# Nginx 리버스 프록시

nginx:

image: nginx:alpine

container\_name: nginx-proxy

depends\_on:

- app

ports:

- "80:80"

- "443:443"

volumes:

- ./nginx.conf:/etc/nginx/nginx.conf

- ./ssl:/etc/nginx/ssl

restart: unless-stopped

volumes:

mysql\_data:

redis\_data:

**3. 환경 변수 파일 (.env)**

# .env 파일 생성

GOOGLE\_CLIENT\_ID=your-google-client-id

GOOGLE\_CLIENT\_SECRET=your-google-client-secret

KAKAO\_CLIENT\_ID=your-kakao-client-id

KAKAO\_CLIENT\_SECRET=your-kakao-client-secret

NAVER\_CLIENT\_ID=your-naver-client-id

NAVER\_CLIENT\_SECRET=your-naver-client-secret

**4. Nginx 설정 (nginx.conf)**

events {

worker\_connections 1024;

}

http {

upstream backend {

server app:8080;

}

# HTTP to HTTPS redirect

server {

listen 80;

server\_name your-domain.com;

return 301 https://$server\_name$request\_uri;

}

# HTTPS server

server {

listen 443 ssl;

server\_name your-domain.com;

ssl\_certificate /etc/nginx/ssl/certificate.crt;

ssl\_certificate\_key /etc/nginx/ssl/private.key;

location / {

proxy\_pass http://backend;

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;

}

# WebSocket 지원

location /ws {

proxy\_pass http://backend;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection "upgrade";

proxy\_set\_header Host $host;

}

}

}

**🚀 CI/CD 파이프라인**

**1. GitHub Actions 설정 (.github/workflows/deploy.yml)**

name: Deploy to AWS EC2

on:

push:

branches: [ main ]

jobs:

deploy:

runs-on: ubuntu-latest

steps:

- name: Checkout code

uses: actions/checkout@v3

- name: Set up JDK 17

uses: actions/setup-java@v3

with:

java-version: '17'

distribution: 'temurin'

- name: Build with Gradle

run: ./gradlew build

- name: Build Docker image

run: docker build -t portfolio-app .

- name: Save Docker image

run: docker save portfolio-app | gzip > portfolio-app.tar.gz

- name: Copy files to EC2

uses: appleboy/scp-action@v0.1.4

with:

host: ${{ secrets.EC2\_HOST }}

username: ubuntu

key: ${{ secrets.EC2\_SSH\_KEY }}

source: "portfolio-app.tar.gz,docker-compose.yml,.env"

target: "/home/ubuntu/portfolio"

- name: Deploy to EC2

uses: appleboy/ssh-action@v0.1.5

with:

host: ${{ secrets.EC2\_HOST }}

username: ubuntu

key: ${{ secrets.EC2\_SSH\_KEY }}

script: |

cd /home/ubuntu/portfolio

docker load < portfolio-app.tar.gz

docker-compose down

docker-compose up -d

docker system prune -f

**2. 수동 배포 스크립트 (deploy.sh)**

#!/bin/bash

echo "🚀 Starting deployment..."

# Git 최신 코드 받기

git pull origin main

# Docker 이미지 빌드

echo "📦 Building Docker image..."

docker-compose build

# 기존 컨테이너 중지 및 제거

echo "🛑 Stopping existing containers..."

docker-compose down

# 새 컨테이너 시작

echo "▶️ Starting new containers..."

docker-compose up -d

# 불필요한 이미지 정리

echo "🧹 Cleaning up..."

docker system prune -f

echo "✅ Deployment completed!"

# 서비스 상태 확인

echo "📊 Service status:"

docker-compose ps

**📋 실행 체크리스트**

**초기 설정**

* [ ] AWS EC2 인스턴스 생성
* [ ] Elastic IP 할당
* [ ] 보안 그룹 설정
* [ ] Docker 설치
* [ ] 도메인 설정 (선택사항)

**소셜 로그인 설정**

* [ ] Google OAuth 클라이언트 생성
* [ ] 카카오 애플리케이션 등록
* [ ] 네이버 애플리케이션 등록
* [ ] 환경 변수 설정

**배포 설정**

* [ ] 프로젝트 파일 업로드
* [ ] Docker Compose 실행
* [ ] SSL 인증서 설정 (Let's Encrypt)
* [ ] 도메인 연결 확인

**테스트**

* [ ] 애플리케이션 접속 확인
* [ ] 소셜 로그인 테스트
* [ ] API 엔드포인트 테스트
* [ ] 데이터베이스 연결 확인

**💡 추가 팁**

**비용 최적화**

# t3.micro (프리티어) 사용 시

- 메모리 제한으로 인한 성능 이슈 가능

- 개발/테스트 용도로는 충분

# 운영 환경 권장사양

- t3.small 이상 (2GB RAM)

- gp3 스토리지 사용

- 스냅샷 자동 백업 설정

**보안 강화**

# SSH 키 기반 인증만 허용

sudo sed -i 's/PasswordAuthentication yes/PasswordAuthentication no/' /etc/ssh/sshd\_config

sudo systemctl restart ssh

# 방화벽 설정

sudo ufw enable

sudo ufw allow ssh

sudo ufw allow 80

sudo ufw allow 443

sudo ufw allow 8080

**모니터링 설정**

# docker-compose.yml에 추가

prometheus:

image: prom/prometheus

ports:

- "9090:9090"

volumes:

- ./prometheus.yml:/etc/prometheus/prometheus.yml

grafana:

image: grafana/grafana

ports:

- "3000:3000"

environment:

- GF\_SECURITY\_ADMIN\_PASSWORD=admin

이제 AWS + Docker + 소셜 로그인 환경이 준비되었습니다! 🎉