# “Insite: 상권분석 플랫폼” Porting Manual

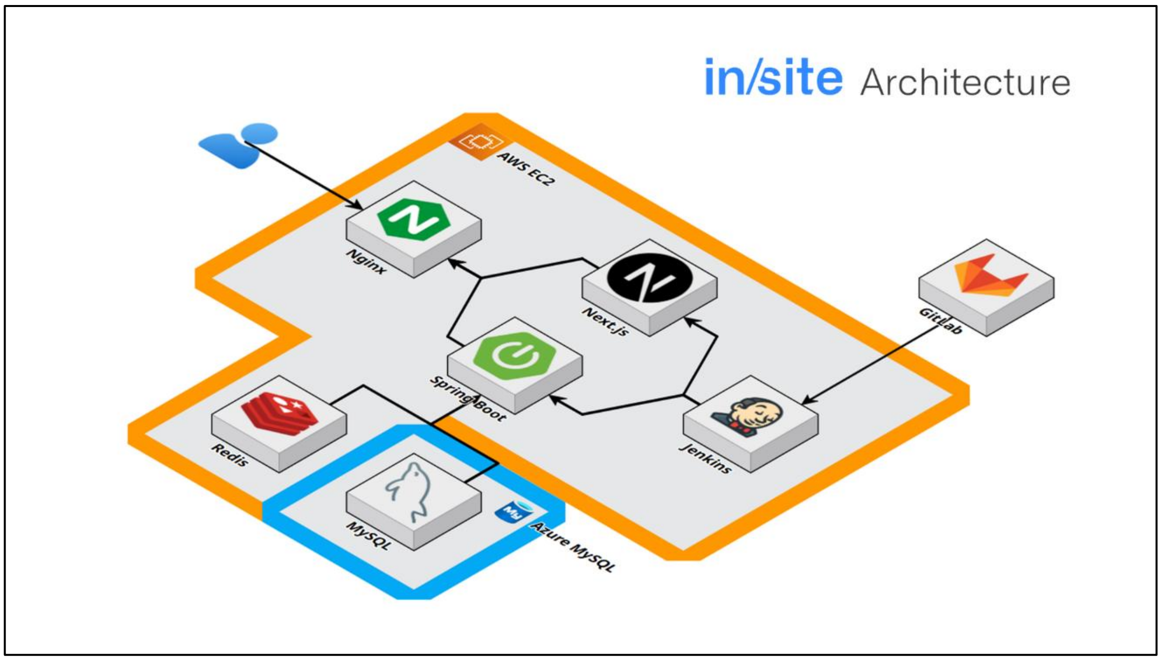
삼성청년SW·AI아카데미

S13P21E203

# 1. 프로젝트 개요

| | |
|---|---|
| **프로젝트명** | Insite: 상권분석 플랫폼 |
| **주요기능** | 인구와 소비 데이터를 포함한 다양한 도시 데이터와 상권 분석 모델을 기반으로 창업자가 최적의 입지를 찾을 수 있도록 상권추천·비교·분석 서비스를 제공하는 플랫폼입니다. |

# 2. 아키텍처 개요



| | |
|---|---|
| **사용기술** | - FE: Next.js(SSR) · TypeScript · Zustand · Axios<br>- BE: Java 17 · Spring Boot 3.5.x · Spring Security · JWT · JPA · jOOQ<br>- DB: MySQL(Azure) · Redis(EC2, Docker)<br>- Infra: AWS EC2(Ubuntu 22.04 LTS) · Nginx(reverse proxy/SSL) · Docker · Jenkins(CI/CD)<br>- AI/Model: HistGradientBoosting(분석 모델) · ChatGPT 4.1 nano(요약/문장 생성 등) |
| **운영흐름** | 1. 사용자는 Nginx(80/443)로 접근<br>2. Nginx → Next.js(3000), Spring Boot(8080) 라우팅<br>3. BE → MySQL(외부/Azure 3306), Redis(6379, EC2내 Docker)<br>4. Jenkins+Redis는 Compose로 영속 볼륨 운영, FE/BE는 Jenkins가 Docker로 빌드/배포<br>5. GitLab → Jenkins(웹훅/토큰) → 컨테이너 재기동 |

## 3. 서버 사전 준비

| | |
|---|---|
| 시스템 요구사항 | - OS: Ubuntu 22.04.4 LTS<br>- Docker: 28.x<br>- Docker Compose v2 플러그인<br>- OpenJDK 17, Node 20(컨테이너 내부에서 사용)<br>- 도메인 + SSL 인증서(Let's Encrypt 권장) |
| Docker/ Compose 설치 | `sudo apt update && sudo apt -y install ca-certificates curl gnupg`<br>`sudo install -m 0755 -d /etc/apt/keyrings`<br>`curl -fsSL https://download.docker.com/linux/ubuntu/gpg \| \`<br>`  sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg`<br>`echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] \`<br>`https://download.docker.com/linux/ubuntu $(. /etc/os-release && echo $VERSION_CODENAME) stable" \| \`<br>`  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null`<br>`sudo apt update`<br>`sudo apt -y install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin`<br>`sudo systemctl enable --now docker`<br>`sudo usermod -aG docker $USER`<br>`# 재로그인 또는: newgrp docker`<br>`docker -v`<br>`docker compose version` |

## 4. 영속 인프라(Jenkins · Redis)

| | |
|---|---|
| 디렉터리/파일 | `mkdir -p ~/infrafoundation/jenkins`<br>`mkdir -p ~/infrafoundation/redis_data`<br>`cd ~/infrafoundation` |
| .env | `# JENKINS`<br>`JENKINS_IMAGE=jenkins/jenkins:lts-jdk17`<br>`JENKINS_HTTP_PORT=8081`<br>`JENKINS_AGENT_PORT=50000`<br>`TZ=Asia/Seoul`<br><br>`# REDIS`<br>`REDIS_PASSWORD=<your_redis_password>` |
| docker-compose.yml | `# FE/BE와 통신을 위해 공유 네트워크(backend-net)를 사용`<br>`# Jenkins가 Docker를 제어하도록 /var/run/docker.sock 마운트` |

```
name: infrafoundation
version: "3.8"

services:
  redis:
    image: redis:7
    container_name: redis
    restart: unless-stopped
    command: ["redis-server", "--requirepass",
"${REDIS_PASSWORD}", "--appendonly", "yes"]
    volumes:
      - ./redis_data:/data
    healthcheck:
      test: ["CMD", "redis-cli", "-a", "${REDIS_PASSWORD}",
"PING"]
      interval: 10s
      timeout: 3s
      retries: 5
    networks: [backend-net]

  jenkins:
    build:
      context: ./jenkins
      dockerfile: Dockerfile
    container_name: jenkins_custom
    restart: unless-stopped
    ports:
      - "${JENKINS_HTTP_PORT}:8080"
      - "${JENKINS_AGENT_PORT}:50000"
    environment:
      - TZ=${TZ}
    volumes:
      - jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
    user: root
    networks: [backend-net]

volumes:
  jenkins_home:

networks:
  backend-net:
    external: true
```

네트워크/
기동

```
docker network inspect backend-net >/dev/null 2>&1 || \
docker network create --driver bridge backend-net

docker compose up -d --build
docker ps
```

## 5. Nginx(리버스 프록시/SSL)

| | |
|---|---|
| **디렉터리 구조** | <pre>/opt/insite/nginx<br>├── conf.d<br>│   ├── insite-http.conf<br>│   ├── insite-ssl.conf<br>│   └── upstreams.conf<br>└── nginx.conf</pre> |
| upstreams.conf | <pre>upstream insite-frontend { server insite-frontend:3000; }<br>upstream insite-backend  { server insite-backend:8080; }</pre> |

insite-ssl.conf

```
server {
    listen 443 ssl;    # ← 'listen ... http2' 제거
    http2 on;          # ← 별도 지시어로 http2 활성화
    server_name <your_domain>;

    ssl_certificate
/etc/letsencrypt/live/<your_domain>/fullchain.pem;
    ssl_certificate_key
/etc/letsencrypt/live/<your_domain>/privkey.pem;

    # FE
    location / {
        proxy_pass http://insite-frontend;
        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;
    }

    # BE
    location /api/ {
        proxy_pass http://insite-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;
    }
}
```

컨테이너 실행예시

```
IMG=nginx:1.27
NET=backend-net
CTN=insite-nginx
CONF_DIR=/opt/insite/nginx
CONF_D=$CONF_DIR/conf.d
LE_DIR=/etc/letsencrypt
CERT_WWW=/var/www/certbot
```

```
docker rm -f "$CTN" || true
docker run -d --name "$CTN" \
  --network "$NET" \
  -p 80:80 -p 443:443 \
  -v "$CONF_D":/etc/nginx/conf.d:ro \
  -v "$CONF_DIR/nginx.conf":/etc/nginx/nginx.conf:ro \
  -v "$CERT_WWW":/var/www/certbot:ro \
  -v "$LE_DIR":/etc/letsencrypt \
  --restart unless-stopped \
  "$IMG"

# 설정 변경 시
docker exec -it insite-nginx nginx -s reload
```

## 6. Jenkins 설정(CI/CD)

**플러그인/크리덴셜**

- GitLab Plugin 설치
- Credentials 2개 필수
  1. GitLab API 토큰 (레포지토리 Access Token)
  2. 개인 Access Token (웹훅/빌드 트리거 등)
- Jenkins 전역 Credentials에 저장 후,
  파이프라인에서 credentialsId로 사용

**FE 파이프라인**

```
pipeline {
  agent any
  options { disableConcurrentBuilds(); timestamps();
timeout(time: 30, unit: 'MINUTES') }

  environment {
    REPO_URL    = 'https://lab.ssafy.com/s13-bigdata-recom-
sub1/S13P21E203.git'
    BRANCH     = 'master'
    NETWORK    = 'backend-net'
    FRONT_CTX  = 'front-end'
    FRONT_IMG  = 'insite-frontend'
    FRONT_CTN  = 'insite-frontend'
    FRONT_PORT = '3000'
    // NEXT_PUBLIC_API_BASE 등 필요시 추가
  }

  stages {
    stage('Git Clone') {
      steps {
        git url: env.REPO_URL, branch: env.BRANCH,
credentialsId: '<gitlab-cred-id>'
      }
    }
    stage('Ensure Network') {
      steps {
        sh """
          docker network inspect ${NETWORK} >/dev/null 2>&1 || \
          docker network create --driver bridge ${NETWORK}
        """
      }
    }
    stage('Build SSR Frontend Image') {
      steps {
        sh """
          set -eu
          docker rm -f ${FRONT_CTN} || true
          docker rmi ${FRONT_IMG} || true
          docker build --pull -t ${FRONT_IMG} ${FRONT_CTX}
        """
      }
    }
    stage('Run SSR Frontend Container') {
      steps {
        sh """
          set -eu
          docker rm -f ${FRONT_CTN} || true
          docker run -d --name ${FRONT_CTN} \
            --network ${NETWORK} \
            --restart unless-stopped \
            -e NODE_ENV=production \
            -e PORT=${FRONT_PORT} \
            ${FRONT_IMG}
          docker ps --filter name=${FRONT_CTN}
        """
      }
    }
    stage('Smoke Check') {
      steps {
        script {
          def attempts = 15; def waitSec = 2; def code = 1
          for (int i = 1; i <= attempts; i++) {
            code = sh(returnStatus: true,
              script: "docker exec ${env.FRONT_CTN} node -e
\"require('http').get('http://127.0.0.1:${env.FRONT_PORT}',
r=>process.exit(r.statusCode<500?0:1)).on('error',()=>process.e
xit(1))\"")
            if (code == 0) { echo "Smoke check passed #${i}";
break }
            sleep time: waitSec, unit: 'SECONDS'
          }
          if (code != 0) {
            sh "docker logs --tail=200 ${env.FRONT_CTN} || true"
            error "Smoke test failed"
          }
        }
        }
      }
    }
  post { success { echo 'SSR frontend running' } failure { echo
'FE deploy failed' } }
}
```

**BE 파이프라인**

```
pipeline {
  agent any
  options { disableConcurrentBuilds() }

  environment {
    BUILD_CONTEXT = 'back-end'
    IMAGE          = 'insite-backend'
    CONTAINER      = 'insite-backend'
    HOST_PORT      = '8080'
    APP_PORT       = '8080'
    NETWORK        = 'backend-net'
  }

  stages {
    stage('Git Clone') {
      steps {
        git url: 'https://lab.ssafy.com/s13-bigdata-recom-
sub1/S13P21E203.git',
            branch: 'master',
            credentialsId: '<gitlab-cred-id>'
      }
    }
    stage('Ensure Network') {
      steps {
        sh """
          docker network inspect ${NETWORK} >/dev/null 2>&1 || \
          docker network create --driver bridge ${NETWORK}
        """
      }
    }
    stage('Check Config (application.yml)') {
      steps {
        sh '''
          docker run --rm -v /opt/insite/config:/config:ro
busybox sh -lc '
            test -f /config/application.yml || { echo "[ERROR]
/config/application.yml not found"; exit 1; }
            echo "[OK] found /config/application.yml"
          '
        '''
      }
    }
    stage('Docker Build') {
      steps {
        sh """
          docker rm -f ${CONTAINER} || true
```

|  |  |
|---|---|
|  | <pre>          docker rmi ${IMAGE} \|\| true<br>          docker build --pull -t ${IMAGE} ${BUILD_CONTEXT}<br>        """<br>      }<br>    }<br>    stage('Run Container') {<br>      steps {<br>        sh """<br>          docker rm -f '${CONTAINER}' \|\| true<br>          docker run -d --name '${CONTAINER}' \\<br>            -p '${HOST_PORT}':'${APP_PORT}' \\<br>            --network '${NETWORK}' \\<br>            --restart unless-stopped \\<br>            -v /opt/insite/config:/config:ro \\<br>            '${IMAGE}' \\<br>            --spring.config.additional-location=/config/<br>        """<br>      }<br>    }<br>  }<br>  post { success { echo 'backend deployed' } failure { echo 'backend failed' } }<br>}</pre> |

## 7. 애플리케이션 설정

| | |
|---|---|
| application.yml | <pre>spring:<br>  datasource:<br>    url: jdbc:mysql://<azure_mysql_host>:3306/<schema>?sslMode=REQUIRED&characterEncoding=UTF-8&serverTimezone=Asia/Seoul<br>    username: <db_user><br>    password: <db_password><br>  jpa:<br>    hibernate:<br>      ddl-auto: none<br>    open-in-view: false<br>  redis:<br>    host: redis      # 같은 네트워크 내 컨테이너 이름<br>    port: 6379<br>    password: ${REDIS_PASSWORD:<redis_password>}<br><br>jwt:<br>  secret: <jwt_secret><br>  access-token-validity-seconds: 3600<br>  refresh-token-validity-seconds: 1209600<br><br>server:<br>  port: 8080</pre> |

## 8. 배포 순서

1. Docker/Compose 설치 → docker -v, docker compose version 확인
2. backend-net 네트워크 생성 → docker network create backend-net
3. infrafoundation(Jenkins/Redis) 디렉터리 배포 → docker compose up -d
4. Jenkins 접속(:8081) → GitLab 플러그인/크리덴셜 설정
5. /opt/insite/config/application.yml 작성
6. FE/BE 파이프라인 생성(멀티브랜치 또는 파이프라인) → 실행
7. Nginx 컨테이너 실행(위 설정) → HTTPS 접속 확인

// END