YourJob 프로젝트 Docker Compose 개발 가이드

이 문서는 **YourJob** 프로젝트에서 **Docker Compose**를 활용하여 개발 환경을 구축하고, **Trunk-Based Development(TBD)** 브랜치 전략을 간단히 소개하며, 실제 개발 사이클을 안내하는 가이드입니다.

목차

개요

- **목표**
    - 개발자가 **Docker Compose**로 **YourJob** 프로젝트의 모든 서비스를(Frontend, BFF, Backend, DB) 로컬에서 띄우고,
    - **Trunk-Based Development (TBD)** 브랜치 전략에 맞춰 원활히 협업할 수 있게 하는 것.
- **구성 요약**
    1. **Frontend**: React (Webpack Dev Server, Hot Reload)
    2. **BFF**: Spring Boot, 포트 8081 (디버그 5007)
    3. **Backend**: Spring Boot, 포트 8082 (디버그 5006), MySQL 8.0 연결
    4. **MySQL DB**: ports: 3307 → 3306
    5. **docker-compose**: 전체 컨테이너 일괄 실행

Docker Compose 기반 개발 환경

프로젝트 구조

```
your-job-repo/
 ├ frontend/
 │   ├ Dockerfile
 │   └ ...
 ├ bff/
 │   ├ Dockerfile
```

```
│   └ ...
├ backend/
│   ├ Dockerfile
│   └ ...
├ docker-compose.yml
└ ...
```

- **frontend/**: React 소스 & Dockerfile
- **bff/**, **backend/**: Spring Boot (Kotlin) & Dockerfile
- **docker-compose.yml**: 이 모든 서비스를 연결하여 로컬에서 실행

docker-compose.yml

아래는 예시입니다. 실제 버전에서는 포트, 환경 변수 등을 수정할 수 있습니다.

```yaml
version: '3.8'

services:
  db:
    image: mysql:8.0
    container_name: yourjob-db
    ports:
      - "3307:3306"
    environment:
      MYSQL_ROOT_PASSWORD: rootpass
      MYSQL_DATABASE: yourjobdb
      MYSQL_USER: yourjob
      MYSQL_PASSWORD: yourjob_pass
    networks:
      - yourjob-network
    volumes:
      - db_data:/var/lib/mysql

  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: yourjob-backend
    ports:
      - "8082:8080"
      - "5006:5005"
    environment:
      SPRING_PROFILES_ACTIVE: local
      DB_HOST: db
      DB_PORT: 3306
      DB_NAME: yourjobdb
      DB_USER: yourjob
      DB_PASSWORD: yourjob_pass
    depends_on:
      - db
    networks:
```

```
      - yourjob-network

  bff:
    build:
      context: ./bff
      dockerfile: Dockerfile
    container_name: yourjob-bff
    ports:
      - "8081:8080"
      - "5007:5005"
    environment:
      SPRING_PROFILES_ACTIVE: local
    depends_on:
      - backend
    networks:
      - yourjob-network

  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
      target: dev        # 개발 모드 스테이지
    container_name: yourjob-frontend
    ports:
      - "3000:3000"
    volumes:
      - ./frontend:/app
      - /app/node_modules
    command: npm start
    networks:
      - yourjob-network

networks:
  yourjob-network:
    driver: bridge

volumes:
  db_data:
```

실행 및 확인

1. **루트 디렉토리**(`your-job-repo/`)에서:

   ```
   docker-compose up --build
   ```

2. 컨테이너가 순차적으로 빌드/실행되며, 로그가 표시됨
3. **확인**:
   - Frontend: http://localhost:3000
   - BFF: http://localhost:8081
   - Backend: http://localhost:8082

- DB: `mysql -h 127.0.0.1 -P 3307 -u yourjob -p`

브랜치 전략: Trunk-Based Development (TBD)

TBD란?

Trunk-Based Development는

- **메인 브랜치(Trunk)** 하나에 자주, 작은 단위로 **merge**하는 방식의 개발 전략입니다.
- 보통 **feature 브랜치**를 짧게 유지하고, 하루~이틀 내로 빠른 단위로 trunk에 병합합니다.
- **Long-lived 브랜치**(오래 유지되는 릴리즈 브랜치 등)를 최소화하여 **지속적 통합(Continuous Integration)**을 강화하는 기법입니다.

개발 사이클(Workflow)

1. **메인 브랜치**: `master` or `main` (Trunk)
2. **단기 feature 브랜치** 생성: `feature/some-new-feature`
   - 최대한 짧은 주기로 작업 (1~2일 이내)
   - **docker-compose** 로컬 개발 환경에서 기능 구현 & 테스트
3. **코드 리뷰 & 머지**:
   - feature 브랜치를 **Trunk**로 Pull Request
   - 단위 테스트 & QA 후 승인
   - 머지 완료 시 feature 브랜치는 바로 삭제
4. **지속적 배포**(CI/CD):
   - Trunk에 merge될 때마다 빌드/테스트/배포 파이프라인이 동작
   - (이 부분은 팀 상황에 따라 설정)

결과:

- 충돌을 최소화하고, 항상 **Trunk**가 **빨리 통합된 최신 코드**를 유지
- 개발자 간 협업이 원활

개발자 사용 가이드 (Step by Step)

1. **레포지토리 클론**

```
git clone https://<repo-url>/your-job-repo.git
cd your-job-repo
```

2. **메인(Trunk) 브랜치 기반 최신화**

```
git checkout main
git pull origin main
```

3. **단기 feature 브랜치 생성**

```
git checkout -b feature/some-new-feature
```

4. **로컬 개발 (docker-compose)**

```
docker-compose up --build
# frontend => http://localhost:3000
# (수정 후 자동 핫로드)
# backend => http://localhost:8082
# bff => http://localhost:8081
```

5. **코드 수정, 커밋 & 푸시**

```
git add .
git commit -m "Add new feature X"
git push origin feature/some-new-feature
```

6. **Pull Request**
   - Git 플랫폼(예: GitHub, GitLab)에서 `feature/some-new-feature` → `main` PR 생성
   - 리뷰 & 피드백 반영
7. **머지 & 브랜치 삭제**
   - 머지가 끝나면 `feature/some-new-feature` 브랜치를 제거
   - `main` 브랜치를 다시 pull 받아 최신화

자주 발생하는 문제

1. **Docker Compose 포트 충돌**
   - 이미 3000/8081/8082/3307 등을 다른 프로세스가 쓰고 있을 수 있음.
   - `docker-compose.yml`에서 포트를 바꿔서 해결
2. **DB Connection Refused**
   - DB가 기동하기 전에 Backend/BFF가 먼저 올라오면 에러
   - 재시도 or healthcheck 도입
3. **브랜치가 오래 살아있어 충돌**
   - TBD에서 권장하는 "짧은 브랜치" 유지 원칙을 지키고, 자주 merge
4. **npm: not found**
   - Frontend Dockerfile에서 **dev** 스테이지 대신 **production** 스테이지 사용하는 경우 발생
   - `docker-compose`에서 `target: dev` 지정 확인

정리

- **Docker Compose**로 **Frontend/BFF/Backend/DB**를 한꺼번에 띄워 **로컬 개발**을 쉽게 진행
- Trunk-Based Development(TBD)로 **메인 브랜치**를 항상 최신 상태로 유지하고, **짧은 feature 브랜치**를 빈번히 머지

- 개발자 워크플로우:
    1. `git checkout -b feature/...`
    2. `docker-compose up --build` 로컬 테스트
    3. 커밋 & Push → Pull Request
    4. 메인 병합 & 브랜치 삭제
- 이를 통해 **지속적인 통합**, **빠른 협업**, **효율적인 CI/CD**가 가능