실행 환경

프로젝트 빌드 및 배포 가이드

1. 개발 환경

1.1 필수 요구사항

항목	버전	용도
Backend (Spring Boot)		
JDK	17+	Core API 실행
Gradle	8.5+	빌드 도구
Backend (FastAPI)		
Python	3.11+	Summary API 실행
pip	Latest	Python 패키지 관리
Frontend		
Node.js	20+	React 개발 서버
npm	10+	패키지 관리
Infrastructure		
Docker	24+	컨테이너 런타임
Docker Compose	2.20+	멀티 컨테이너 관리
Database		
PostgreSQL	16+	메인 데이터베이스
Redis	7+	캐시 서버

1.2 권장 개발 도구

- **IDE**: IntelliJ IDEA (Backend), VS Code (Frontend/Python)
- **API 테스트**: Postman, HTTPie
- **Git Client**: GitKraken, SourceTree
- **Database Client**: DBeaver, pgAdmin

---

2. 빌드 시 사용되는 환경 변수

2.1 공통 환경변수

환경변수	용도	예시 값	필수 여부
`SPRING_PROFILES_ACTIVE`	Spring Boot 프로파일 선택	`local` , `docker` , `prod`	✅ 필수
`GMS_BASE_URL`	GMS API 엔드포인트	`https://gms.ssafy.io/gmsapi/`	✅ 필수
`TTIBU_CRYPTO_SECRET`	암호화 키 (32자 이상)	`ttibu-$$yeahjiyeongtaegyu`	✅ 필수

2.2 데이터베이스 환경변수

환경변수	용도	로컬 개발	Docker	AWS 배포
`DB_URL`	PostgreSQL JDBC URL	`jdbc:postgresql://localhost:5432/ttibu_db`	`jdbc:postgresql://postgres:5432/ttibu_db`	`jdbc:postgresql://<RDS_ENDPOINT>:5432/ttibu_db`
`DB_USERNAME`	DB 사용자명	`postgres`	`postgres`	`postgres`
`DB_PASSWORD`	DB 비밀번호	`postgres`	`postgres`	`<RDS_PASSWORD>`
`REDIS_HOST`	Redis 호스트	`localhost`	`redis`	`<ELASTICACHE_ENDPOINT>`
`REDIS_PORT`	Redis 포트	`6379`	`6379`	`6379`

2.3 LiteLLM 환경변수

환경변수	용도	예시 값	필수 여부
`LITELLM_BASE_URL`	LiteLLM 프록시 URL	`http://localhost:4000` (로컬) `http://litellm:4000` (Docker)	✅ 필수
`LITELLM_MASTER_KEY`	LiteLLM 인증 키	`sk-1234`	✅ 필수
`LITELLM_MODEL`	기본 사용 모델	`gpt-4o-mini`	✅ 필수
`LITELLM_CONFIG_PATH`	설정 파일 경로	`file:/app/litellm-config.yaml`	❌ 선택
`LITELLM_POLL_MS`	설정 폴링 주기 (ms)	`30000`	❌ 선택

2.4 Summary API 환경변수

환경변수	용도	예시 값	필수 여부
`SUMMARY_API_BASE_URL`	Summary API URL	`http://localhost:8001` (로컬) `http://summary-api:8001` (Docker)	✅ 필수
`SUMMARY_API_TIMEOUT_MS`	API 타임아웃 (ms)	`30000`	❌ 선택 (기본값: 10000)

2.5 AWS 배포용 환경변수 (추가)

환경변수	용도	예시 값	필수 여부
`ECR_REGISTRY`	ECR Public 레지스트리	`public.ecr.aws/v8n3v4s8`	✅ 필수
`RDS_ENDPOINT`	RDS 엔드포인트	`ttibu-postgres.xxxxx.ap-northeast-2.rds.amazonaws.com`	✅ 필수
`RDS_DB_NAME`	RDS 데이터베이스명	`ttibu_db`	✅ 필수
`RDS_USERNAME`	RDS 사용자명	`postgres`	✅ 필수
`RDS_PASSWORD`	RDS 비밀번호	`<secure-password>`	✅ 필수
`REDIS_ENDPOINT`	ElastiCache 엔드포인트	`ttibu-redis.xxxxx.cache.amazonaws.com`	✅ 필수
`S3_CONFIG_BUCKET`	S3 설정 버킷명	`s3-litellm-config`	❌ 선택 (기본값 설정됨)

3. 배포 시 특이사항

3.1 DDL 자동 생성 주의

⚠️ **중요**: 환경별 DDL 설정이 다릅니다.

프로파일	`spring.jpa.hibernate.ddl-auto`	설명
`local`	`update`	테이블 자동 생성/수정
`docker`	`update`	테이블 자동 생성/수정
`prod`	`validate`	스키마 검증만 (변경 불가) ⚠️

프로덕션 배포 시:

- DDL은 **수동으로 실행** 필요
- Flyway/Liquibase 마이그레이션 도구 사용 권장
- 초기 배포 시 스키마 파일 직접 실행: `schema.sql` (있는 경우)

3.2 Docker Container 의존성

컨테이너 시작 순서:

```
1. postgres (데이터베이스)
   ↓
2. redis (캐시)
   ↓
3. litellm (AI 프록시)
   summary-api (요약 서비스)
   ↓
4. core-api (메인 백엔드)
   ↓
5. frontend (프론트엔드)
```

depends_on 설정:

- `core-api`: postgres, redis, litellm, summary-api에 의존
- `frontend`: core-api에 의존

주의사항:

- `depends_on`은 **시작 순서만 보장** (준비 완료 X)
- **healthcheck** 설정으로 서비스 준비 상태 확인
- Summary API는 **모델 로딩으로 시작 시간 3-5분 소요**

3.3 포트 번호

사용 중인 포트 (충돌 주의):

서비스	포트	용도
Frontend	80	Nginx 웹 서버 (로컬: 3000)
Core API	8080	Spring Boot REST API
Summary API	8001	FastAPI 요약 서비스
LiteLLM	4000	AI 모델 프록시
PostgreSQL	5432	데이터베이스

서비스	포트	용도
Redis	6379	캐시 서버

포트 충돌 확인:

```
# Linux/Mac
lsof -i :8080

# Windows
netstat -ano | findstr :8080
```

3.4 Docker 네트워크

네트워크 설정:

- **이름:** `ttibu-network`
- **드라이버:** `bridge`
- **격리:** 모든 컨테이너가 동일 네트워크 내부에서 통신

내부 통신:

- Frontend → Core API: `http://core-api:8080/api/*` (Nginx proxy)
- Core API → LiteLLM: `http://litellm:4000`
- Core API → Summary API: `http://summary-api:8001`
- Core API → Redis: `redis:6379`
- Core API → PostgreSQL: `postgres:5432`

외부 접근:

- 개발 환경: `localhost:<포트>`
- AWS 배포: ALB DNS → EC2 → Frontend (80)

3.5 볼륨 마운트 주의사항

영구 저장 데이터:

```
volumes:
  postgres_data:  # PostgreSQL 데이터 (영구 저장)
  redis_data:     # Redis 스냅샷 (AOF 모드)
```

설정 파일 마운트:

- `litellm-config.yaml`:
  - 로컬: `./litellm-config.yaml:/app/config.yaml:ro`
  - AWS: `/home/ubuntu/litellm-config.yaml:/app/config.yaml:ro`

⚠️ **주의:** AWS 배포 시 S3에서 자동 다운로드 (`before_install.sh`)

---

4. 주요 계정 및 프로퍼티 파일 목록

4.1 환경 설정 파일

파일 경로	용도	버전 관리
`.env`	로컬 개발 환경변수	❌ `.gitignore` (보안)
`.env.example`	환경변수 템플릿	✅ Git 포함
`.env.prod`	AWS 배포용 환경변수	❌ `.gitignore` (보안)
`.env.prod.example`	배포용 템플릿	✅ Git 포함

4.2 Spring Boot 프로퍼티

파일 경로	프로파일	용도
`core-api/src/main/resources/application.yml`	공통	기본 설정
`core-api/src/main/resources/application-local.yml`	`local`	로컬 개발
`core-api/src/main/resources/application-docker.yml`	`docker`	Docker Compose
`core-api/src/main/resources/application-prod.yml`	`prod`	AWS 프로덕션

4.3 Docker Compose 파일

파일 경로	용도
`docker-compose.yml`	로컬 개발 (PostgreSQL, Redis 포함)
`docker-compose.prod.yml`	AWS 배포 (RDS, ElastiCache 연동)

4.4 LiteLLM 설정

파일 경로	용도	저장 위치
`litellm-config.yaml`	AI 모델 설정	로컬: 프로젝트 루트 AWS: S3 (`s3-litellm-config`)

주요 설정 내용:

- GPT, Claude, Gemini 모델 목록
- `master_key`: LiteLLM 인증 키
- 프록시 설정

4.5 데이터베이스 계정

PostgreSQL 기본 계정 (로컬/Docker):

```
사용자명: postgres
비밀번호: postgres
데이터베이스: ttibu_db
```

AWS RDS 계정 (프로덕션):

```
사용자명: postgres
비밀번호: <RDS_PASSWORD> (환경변수)
데이터베이스: ttibu_db
엔드포인트: <RDS_ENDPOINT> (환경변수)
```

4.6 외부 API 키 (필요 시)

LiteLLM을 통해 사용하는 AI 모델:

- OpenAI API Key: 환경변수 `OPENAI_API_KEY`
- Anthropic API Key: 환경변수 `ANTHROPIC_API_KEY`
- Google AI Key: 환경변수 `GOOGLE_API_KEY`

⚠️ **주의**: API 키는 `.env` 파일에 저장하고 **Git에 커밋 금지**

---

5. 개발 환경 테스트 가이드 (Docker Compose)

5.1 포함된 서비스

docker-compose.yml에 포함된 서비스:

서비스	이미지/빌드	포트	설명
메인 애플리케이션			
`core-api`	Spring Boot (Gradle 빌드)	8080	REST API 서버
`frontend`	React (Vite + Nginx)	80	웹 프론트엔드
`summary-api`	FastAPI (Python 3.11)	8001	AI 요약 서비스
`litellm`	`ghcr.io/berriai/litellm:main-latest`	4000	AI 모델 프록시
데이터베이스			
`postgres`	PostgreSQL 16 (커스텀 Dockerfile)	5432	메인 데이터베이스
`redis`	Redis 7 Alpine	6379	캐시 서버

의존성 관계:

```
postgres ──┬── core-api ── frontend
redis ─────┤
litellm ───┤
summary-api┘
```

5.2 실행 방법

5.2.1 전체 환경 실행

```
# 1. 환경변수 파일 생성 (최초 1회)
cp .env.example .env
# .env 파일 편집 (GMS_BASE_URL, API 키 등)

# 2. 전체 서비스 시작
docker-compose up -d

# 3. 로그 확인
docker-compose logs -f

# 4. 서비스 중지
docker-compose down

# 5. 볼륨까지 삭제 (데이터 초기화)
docker-compose down -v
```

빌드 포함 실행 (코드 변경 시):

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

특정 서비스만 재빌드:

```
docker-compose up -d --build core-api
```

5.2.2 인프라만 실행 (권장 개발 방식)

Backend/Frontend를 로컬에서 개발하고, DB/Redis만 Docker 사용:

```
# PostgreSQL + Redis만 실행
docker-compose up -d postgres redis litellm

# 확인
docker-compose ps
```

이후 로컬에서:

```
# Backend (Spring Boot)
cd core-api
./gradlew bootRun --args='--spring.profiles.active=local'

# Frontend (React)
cd frontend
npm run dev

# Summary API (FastAPI)
cd summary-api
uvicorn app.main:app --reload --host 0.0.0.0 --port 8001
```

5.2.3 서비스 상태 확인

```
# 컨테이너 상태
docker-compose ps

# 헬스체크 상태 포함
docker ps

# 특정 서비스 로그
docker-compose logs core-api
docker-compose logs -f summary-api  # 실시간

# 리소스 사용률
docker stats
```

정상 상태 예시:

```
NAME              STATUS                    PORTS
ttibu-frontend    Up 2 minutes (healthy)    0.0.0.0:80→80/tcp
```

```
ttibu-core-api     Up 2 minutes (healthy)    0.0.0.0:8080→8080/tcp
ttibu-summary-api  Up 5 minutes (healthy)    0.0.0.0:8001→8001/tcp
ttibu-litellm      Up 5 minutes (healthy)    0.0.0.0:4000→4000/tcp
ttibu-postgres     Up 5 minutes              0.0.0.0:5432→5432/tcp
ttibu-redis        Up 5 minutes              0.0.0.0:6379→6379/tcp
```

5.2.4 접근 URL

서비스	URL	설명
Frontend	http://localhost	웹 애플리케이션 메인
Core API	http://localhost:8080	REST API 엔드포인트
- Swagger UI	http://localhost:8080/swagger-ui.html	API 문서 (로컬만)
- Actuator	http://localhost:8080/actuator/health	헬스체크
Summary API	http://localhost:8001	요약 API
- Docs	http://localhost:8001/docs	FastAPI 문서
LiteLLM	http://localhost:4000	AI 프록시
PostgreSQL	`localhost:5432`	DB 연결 (DBeaver 등)
Redis	`localhost:6379`	Redis CLI 연결

5.3 데이터베이스 설정

5.3.1 PostgreSQL 접속 정보

```
호스트: localhost
포트: 5432
데이터베이스: ttibu_db
사용자명: postgres
비밀번호: postgres
```

DBeaver 연결 예시:

```
Driver: PostgreSQL
Host: localhost
Port: 5432
Database: ttibu_db
Username: postgres
Password: postgres
```

5.3.2 초기 스키마

DDL 자동 생성 모드 (`local`, `docker` 프로파일):

- `spring.jpa.hibernate.ddl-auto=update`
- 첫 실행 시 자동으로 테이블 생성

수동 스키마 실행 (필요 시):

```
# PostgreSQL 컨테이너 접속
docker exec -it ttibu-postgres psql -U postgres -d ttibu_db

# SQL 파일 실행
docker exec -i ttibu-postgres psql -U postgres -d ttibu_db < schema.sql
```

5.3.3 데이터베이스 초기화

```
# 볼륨 삭제 (모든 데이터 삭제)
docker-compose down -v

# 재시작 (깨끗한 상태)
docker-compose up -d postgres
```

---

6. 개발 워크플로우

6.1 권장 개발 방식

Option 1: 인프라만 Docker (권장)

장점:

- 빠른 코드 수정 및 테스트
- IDE 디버깅 가능
- Hot Reload 지원

실행 순서:

```
# 1. 인프라 시작
docker-compose up -d postgres redis litellm

# 2. Backend 로컬 실행
cd core-api
./gradlew bootRun --args='--spring.profiles.active=local'

# 3. Frontend 로컬 실행 (다른 터미널)
cd frontend
npm run dev

# 4. Summary API 로컬 실행 (선택)
cd summary-api
pip install -r requirements.txt
uvicorn app.main:app --reload --port 8001
```

환경변수 설정 (`.env` 또는 IDE Run Configuration):

```
SPRING_PROFILES_ACTIVE=local
DB_URL=jdbc:postgresql://localhost:5432/ttibu_db
REDIS_HOST=localhost
LITELLM_BASE_URL=http://localhost:4000
SUMMARY_API_BASE_URL=http://localhost:8001
```

Option 2: 전체 Docker (통합 테스트)

장점:

- 프로덕션 환경과 동일
- 의존성 격리
- 배포 전 통합 테스트

실행 순서:

```
# 1. 전체 빌드 및 시작
docker-compose up -d --build

# 2. 로그 모니터링
docker-compose logs -f core-api frontend

# 3. 코드 변경 시 특정 서비스만 재빌드
docker-compose up -d --build core-api
```

6.2 환경 관리

6.2.1 프로파일별 전환

로컬 개발:

```
export SPRING_PROFILES_ACTIVE=local
./gradlew bootRun
```

Docker 환경:

```
# docker-compose.yml
environment:
  SPRING_PROFILES_ACTIVE: docker
```

AWS 배포:

```
# docker-compose.prod.yml
environment:
  SPRING_PROFILES_ACTIVE: prod
```

6.2.2 환경변수 우선순위

1. **시스템 환경변수** (최우선)
2. `.env` **파일** (Docker Compose)
3. **application-{profile}.yml** (Spring Boot 기본값)

6.2.3 민감 정보 관리

⚠️ Git에 커밋 금지:

- `.env`
- `.env.prod`
- `application-secret.yml` (있는 경우)

안전한 관리 방법:

1. **로컬**: `.env.example` 복사 후 값 입력
2. **AWS**: GitHub Secrets + CodeDeploy 환경변수
3. **팀 공유**: 1Password, AWS Secrets Manager

---

7. 배포 환경 가이드

7.1 배포 아키텍처

AWS 멀티 AZ 구성:

```
Internet
  ↓
Application Load Balancer (Public Subnet A/C)
  ↓
EC2 Instance (Private Subnet A)
  - Docker Compose (docker-compose.prod.yml)
  - Frontend (Nginx)
  - Core API (Spring Boot)
  - Summary API (FastAPI)
  - LiteLLM Proxy
  ↓
RDS PostgreSQL (Multi-AZ)
ElastiCache Redis (Single-AZ)
S3 (litellm-config.yaml)
```

7.2 배포 프로세스 (CI/CD)

GitHub Actions → ECR → CodeDeploy → EC2:

```
1. 코드 Push (main 브랜치)
  ↓
2. GitHub Actions 수동 트리거
  - Service 선택 (all / frontend / core-api / summary-api)
  - Deploy 선택 (yes / no)
  ↓
3. Docker 이미지 빌드
  ↓
4. ECR Public에 푸시
  - public.ecr.aws/v8n3v4s8/ttibu/frontend:latest
  - public.ecr.aws/v8n3v4s8/ttibu/core-api:latest
  - public.ecr.aws/v8n3v4s8/ttibu/summary-api:latest
  ↓
5. S3에 배포 패키지 업로드
  - docker-compose.prod.yml
  - appspec.yml
  - scripts/ (배포 스크립트)
  - litellm-config.yaml
  ↓
```

```
6. CodeDeploy 배포 트리거
   ↓
7. EC2에서 배포 스크립트 실행
   - ApplicationStop: 기존 컨테이너 중지
   - BeforeInstall: S3 config 다운로드, ECR 로그인
   - ApplicationStart: docker-compose pull & up
   - ValidateService: 헬스체크 (최대 15분)
```

7.3 수동 배포 가이드

7.3.1 GitHub Actions 수동 실행

```
1. GitHub Repository → Actions 탭
2. "Build and Push to ECR Public" 워크플로우 선택
3. "Run workflow" 클릭
4. 옵션 선택:
   - Service: all (전체 빌드)
   - Deploy: yes (EC2 배포)
5. "Run workflow" 버튼 클릭
```

7.3.2 EC2 직접 배포

SSH 접속:

```
ssh -i your-key.pem ubuntu@<ec2-ip>
```

배포 명령어:

```
# 1. 애플리케이션 디렉토리로 이동
cd /home/ubuntu/ttibu-app

# 2. ECR Public 로그인
aws ecr-public get-login-password --region us-east-1 | \
  docker login --username AWS --password-stdin public.ecr.aws

# 3. 최신 이미지 pull
docker-compose -f docker-compose.prod.yml --env-file .env pull

# 4. 컨테이너 재시작
docker-compose -f docker-compose.prod.yml --env-file .env up -d

# 5. 상태 확인
docker-compose -f docker-compose.prod.yml ps
docker-compose -f docker-compose.prod.yml logs -f
```

7.4 배포 후 확인사항

7.4.1 헬스체크

```
# Frontend
curl http://localhost:80

# Core API
curl http://localhost:8080/actuator/health

# Summary API
curl http://localhost:8001/health

# LiteLLM
curl http://localhost:4000/health
```

7.4.2 ALB를 통한 외부 접근

```
# ALB DNS 확인
aws elbv2 describe-load-balancers \
  --names ttibu-alb \
  --query 'LoadBalancers[0].DNSName' \
```

```
  --output text

# 접근 테스트
curl http://<alb-dns-name>
```

7.4.3 로그 확인

```
# 전체 로그
docker-compose -f docker-compose.prod.yml logs

# 특정 서비스 로그
docker logs ttibu-core-api --tail 100

# 실시간 로그
docker logs -f ttibu-summary-api
```

7.5 롤백 방법

Option 1: 이전 배포로 롤백

```
# AWS Console
CodeDeploy → Deployments → 이전 성공한 배포 선택 → Redeploy
```

Option 2: 특정 이미지 버전으로 롤백

```
# docker-compose.prod.yml 수정
image: ${ECR_REGISTRY}/ttibu/core-api:sha-abc123  # 이전 SHA 태그

# 재배포
docker-compose -f docker-compose.prod.yml up -d core-api
```

7.6 트러블슈팅

문제 1: 컨테이너 시작 실패

```
# 로그 확인
docker logs ttibu-core-api

# 일반적인 원인:
# - .env 파일 누락
# - RDS/Redis 연결 실패
# - 환경변수 오타
```

해결:

```
# .env 파일 확인
cat /home/ubuntu/ttibu-app/.env

# RDS 연결 테스트
psql -h <RDS_ENDPOINT> -U postgres -d ttibu_db
```

문제 2: Summary API 메모리 부족

```
# 리소스 사용률 확인
docker stats

# EC2 메모리 확인
free -h
```

해결:

- EC2 인스턴스 타입 업그레이드 (t3.medium → t3.large)

문제 3: ECR 이미지 pull 실패

```
# IAM Role 권한 확인
aws sts get-caller-identity
```

```
# ECR 로그인 재시도
aws ecr-public get-login-password --region us-east-1 | \
  docker login --username AWS --password-stdin public.ecr.aws
```

7.7 모니터링

권장 모니터링 항목:

- **CloudWatch Logs**: 컨테이너 로그 수집
- **CloudWatch Metrics**: CPU, 메모리, 네트워크
- **ALB Metrics**: Request count, Latency, 5XX errors
- **RDS Metrics**: Connection count, CPU, Storage

간단한 헬스체크 스크립트:

```
#!/bin/bash
# health-check.sh

echo "=== TTIBU Health Check ==="
curl -sf http://localhost:80 && echo "✓ Frontend OK" || echo "× Frontend Failed"
curl -sf http://localhost:8080/actuator/health && echo "✓ Core API OK" || echo "× Core API Failed"
curl -sf http://localhost:8001/health && echo "✓ Summary API OK" || echo "× Summary API Failed"
curl -sf http://localhost:4000/health && echo "✓ LiteLLM OK" || echo "× LiteLLM Failed"
```

---

8. 참고 자료

8.1 문서

- Spring Boot 공식 문서
- FastAPI 공식 문서
- Docker Compose 문서
- AWS CodeDeploy 가이드
- LiteLLM 문서

8.2 관련 파일

- `README.md`: 프로젝트 개요
- `docs/AWS_DEPLOYMENT.md`: AWS 인프라 구축 가이드
- `.env.example`: 환경변수 템플릿
- `appspec.yml`: CodeDeploy 배포 설정
- `scripts/`: 배포 스크립트 모음