



포팅 매뉴얼

저희 팀은 Jenkins CI/CD를 이용하여 자동화 배포를 구현하였으나, 해당 문서에서는 Jenkins를 사용하지 않고 수동으로 프로젝트를 포팅하는 방법에 대하여 설명합니다.

시스템 요구사항

저희 서비스는 EC2를 통해 배포하였으며, 다음 사양의 인스턴스에서 무리 없이 구동되었습니다.

- 4 Core CPU
- 16GB RAM

또한, 저희 서비스는 도커를 기반으로 배포를 진행하여 다음 패키지와 환경이 구성되어야 합니다.

- Ubuntu
- Docker
- Docker Compose

서비스 구성

저희 팀의 서비스 타닥이 성공적으로 구동되기 위해서는 다음의 인스턴스들이 필수적으로 구동되어야 합니다.

Storage

- Minio
 - 정적 파일(이미지)을 저장하는 오픈소스 스토리지

Database

- MySQL-Main
 - 서비스 Database
- MySQL-User
 - MSA 회원 게이트웨이 전용 DB
 - 유저 정보 DB
- REDIS
 - 조회수 업데이트 캐싱
 - 챗봇 대화 내역 저장
- MongoDB
 - 상품 정보 데이터베이스
- Qdrant
 - RAG를 위한 상품 벡터 DB

Backend

- Caddy
 - 리버스 프록싱
 - 보안 연결

- Ktor 인증 게이트 웨이
- Spring API 서버
- FastAPI 챗봇 서버

Frontend

- React 프론트 서버 (Nginx)

GitLab Clone

원활한 포팅을 위해 소스 코드를 클론합니다.

```
git clone https://lab.ssafy.com/s12-final/S12P31A703.git
```

Database/Storage 포팅

서비스에 필요한 데이터베이스의 경우 Docker Compose를 통해 간단하게 컨테이너를 띄울 수 있습니다. Database를 위해 **두 개의 Docker Compose 파일이 준비되어 있어, 반드시 둘 다 실행해야 서비스가 원활히 동작할 수 있습니다.**

첫 번째 Docker Compose






해당 파일은 인증 게이트웨이에서 사용하는 User DB를 제외한 모든 DB를 띄우는 데 사용합니다. 파일의 위치는 프로젝트 최상단 경로에 존재하여 다음 명령을 수행하면 실행할 수 있습니다.

```
docker compose up -d
```

해당 Compose 파일을 정상적으로 구동하기 위해선 **적절한 환경 변수 값을 설정**하여야 합니다.

같은 경로에 **.env** 파일을 작성하여 Compose 파일이 정상 동작할 수 있도록 조정해야 합니다.

위의 설정으로 띄워지는 컨테이너는 다음과 같습니다.

- Minio 
- MySQL-Main 
- REDIS 
- MongoDB 
- Qdrant 

두 번째 Docker Compose

다음은 회원 인증 게이트웨이에서 사용하는 DB를 포팅해야 합니다. 해당 Compose 파일의 경우는 **/infra/auth** 에 위치하고 있습니다. **해당 위치로 이동하는 명령은 다음과 같습니다.**

```
cd infra/auth
```

성공적으로 경로를 이동하였다면, 다음 명령을 수행하여 데이터 베이스를 띄웁니다.

```
docker compose up -d
```

위의 설정으로 띄워지는 컨테이너는 다음과 같습니다.

- MySQL-User

덤프 파일 적용

서비스가 정상적으로 실행되기 위해선, DB에 덤프가 적용되어야 동작합니다. 다음의 파일을 다운 받은 후 각각 맞는 DB에 덤프를 적용해주시길 바랍니다.

Backend 포팅

타닥의 백엔드는 총 4가지로 구성되어 있습니다. 모든 서비스를 띄워야 정상적으로 서비스 구동이 가능합니다.

Caddy

caddy는 nginx와 유사한 기능을 수행하는 오픈소스 웹 서버입니다. 저희 서비스는 caddy를 이용하여 모든 요청을 프록싱하고 있습니다.

caddy 역시도 docker compose를 이용해 간단하게 띄울 수 있습니다. `/infra/caddy` 경로로 이동하여 다음 명령을 수행합니다.

```
cd infra/caddy
docker compose up -d
```

위의 설정으로 띄워지는 컨테이너는 다음과 같습니다.

- Caddy

회원 인증 인가 게이트웨이

저희 서비스는 인증을 위한 게이트웨이를 사용하고 있습니다. 모든 백엔드 요청은 Caddy에 의하여 게이트웨이로 프록싱되며 게이트웨이에서는 해당 요청이 인증된 요청인지 검증을 수행합니다.

1. 정상적인 포팅을 위해 `/server/auth-gateway/src/resources` 경로로 이동하여 설정 파일을 생성합니다.

[illegible]

```
- pathPrefix: "/qdrant"
  targetUrl: "http://td-fastapi-dev:8080"
```

minio:

```
url: "https://minio.tadak.kr"
accessKey: "dadada"
secretKey: "dadada!!"
bucket: "image"
baseUrl: "https://minio.tadak.kr"
```

oauth:

kakao:

```
client-id: "1a3c84e2a5568d5e18f196334f194d36"
token-uri: "https://kauth.kakao.com/oauth/token"
user-uri: "https://kapi.kakao.com/v2/user/me"
dev-redirect-uri: "https://dev.tadak.kr/auth/kakao/callback"
local-redirect-uri: "http://localhost:5173/auth/kakao/callback"
deploy-redirect-uri: "https://tadak.kr/auth/kakao/callback"
```

naver:

```
client-id: "RTbsKWEgeiPP2we3tk0x"
client-secret: "rY_TQ6ITVy"
token-uri: "https://nid.naver.com/oauth2.0/token"
user-uri: "https://openapi.naver.com/v1/nid/me"
dev-redirect-uri: "https://dev.tadak.kr/auth/naver/callback"
local-redirect-uri: "http://localhost:5173/auth/naver/callback"
deploy-redirect-uri: "https://tadak.kr/auth/naver/callback"
```

2. `/server/auth-gateway` 경로로 이동한 후 다음 명령을 통해 소스 코드를 빌드한 후 도커 이미지를 실행합니다.

```
cd server/auth-gateway
```

```
docker build -t td-ktor-image-dev
```

```
docker run -d --name td-ktor-dev --network caddy-dev-net td-ktor-image-dev
```

해당 게이트웨이는 외부로 노출하지 않고 반드시 **Caddy**를 통해 프록싱되어 접근되어야 합니다. 즉, **Docker Network**를 적절히 구성하여 Caddy를 통해서만 접근 가능하도록 설정하여야 합니다.

위의 설정으로 띄워지는 컨테이너는 다음과 같습니다.

- td-ktor-dev 

Spring API 서버

서비스의 실질적인 API를 구현하고 기능하는 서버는 **Spring API** 서버입니다. 해당 서비스는 타닥의 핵심이며, 다른 것은 없어도 이 것은 반드시 띄워져야 합니다. 소스 코드의 경로는 `/server/spring` 입니다.

1. `/server/spring/src/main/resources/application.yaml` 경로로 이동하여 설정 파일인 `application.yaml` 파일을 생성합니다.

```
spring:
  datasource:
    url: jdbc:mysql://k12a703.p.ssafy.io:3307/tadak_dev
```

```

username: root
password: dadadada@
driver-class-name: com.mysql.cj.jdbc.Driver

user-datasource:
  url: jdbc:mysql://k12a703.p.ssafy.io:3310/tadakuser?serverTimezone=Asia/Seoul
  driver-class-name: com.mysql.cj.jdbc.Driver
  username: dadadareadonly
  password: dadadada!!

data:
  mongodb:
    username: dadada
    password: dadadada@
    host: k12a703.p.ssafy.io
    port: 27017
    database: danawa
    authentication-database: admin

  redis:
    host: tadak.kr
    port: 6379
    password: dadadada@
    repositories:
      enabled: true

jpa:
  hibernate:
    ddl-auto: update
  properties:
    hibernate:
      format_sql: true
    show-sql: true

portone.secret:
  api-key: 2y5ZOISolCwgBmWvGj1a2dVr0P45Nh0UZRSYghzKuyfxZDfNN1QoyEzeXMfOskUnuacVsk7IAoVF766j
  webhook-key: whsec_7UKGug37FOUcXLgiCvIWIGekVVZf2lh9gFjDbF9LjOQ=

minio:
  endpoint: https://minio.tadak.kr
  access-key: dadada
  secret-key: dadada!!
  public-bucket: image
  private-bucket: private-source

mattermost:
  webhook-url: "https://meeting.ssafy.com/hooks/c3ssooi9ipdjxyk651trdipisy"

```

2. `/server/spring` 경로로 이동한 후 인증 게이트웨이와 마찬가지로 **소스 코드를 빌드한 후 도커 이미지를 실행**합니다.

```

cd server/spring

docker build -t td-spring-image-dev

docker run -d --name td-spring-dev --network caddy-dev-net td-spring-image-dev

```

해당 서비스는 외부로 노출하지 않고 반드시 인증 게이트웨이를 통해 프록싱되어 접근되어야 합니다. 즉, **Docker Network**를 적절히 구성하여 인증 게이트웨이를 통해서만 접근 가능하도록 설정하여야 합니다.

위의 설정으로 띄워지는 컨테이너는 다음과 같습니다.

- td-spring-dev 

FastAPI 챗봇 서버

서비스의 기능 중 하나인 챗봇 기능은 FastAPI 서버를 통해 기능합니다. 해당 서버가 포팅되지 않으면 챗봇 기능이 동작하지 않습니다. 소스 코드의 경로는 `/server/fastapi` 입니다.

1. `/server/fastapi` 경로로 이동한 후 `.env` 파일을 생성합니다.

```
QDRANT_URL="tadak-qdrant-deploy"
QDRANT_PORT=6333

REDIS_HOST="k12a703.p.ssafy.io"
REDIS_PORT=6379
REDIS_PASSWORD="dadadada@"

GOOGLE_API_KEY=AlzaSyDuHGRwP3ExoYUcfqnlADbXNu_n1pditBc
```

2. `/server/fastapi` 폴더에서 다음 명령을 수행한 후 소스 코드를 빌드 후 실행합니다.

```
cd server/fastapi

docker build -t td-fastapi-image-dev

docker run -d --name td-fastapi-dev --network caddy-dev-net td-fastapi-image-dev
```

해당 서비스는 외부로 노출하지 않고 반드시 인증 게이트웨이를 통해 프록싱되어 접근되어야 합니다. 즉, **Docker Network**를 적절히 구성하여 인증 게이트웨이를 통해서만 접근 가능하도록 설정하여야 합니다.

위의 설정으로 띄워지는 컨테이너는 다음과 같습니다.

- td-fastapi-dev 

Frontend 포팅

포팅의 마지막 관문인 프론트엔드 포팅입니다. 프론트엔드 코드의 경우 `/client/web` 경로에 위치합니다.

1. `/client/web` 폴더에 `.env` 파일을 생성합니다.

```
VITE_API_BASE_URL=https://dapi.tadak.kr
VITE_KAKAO_REDIRECT_URI=https://dev.tadak.kr/auth/kakao/callback
VITE_KAKAO_REST_API_KEY=1a3c84e2a5568d5e18f196334f194d36
VITE_KAKAO_JAVASCRIPT_KEY=55c425c1c4155074542cb40b146fbdaa
VITE_SHARE_URL=https://dev.tadak.kr
```

```
VITE_NAVER_REDIRECT_URI=https://dev.tadak.kr/auth/naver/callback
VITE_NAVER_REST_API_KEY=RTbsKWEGeiPP2we3tk0x
VITE_ENVIRONMENT=dev
```

2. `/client/web` 폴더에서 다음 명령을 수행한 후 **소스 코드를 빌드 후 실행**합니다.

```
cd client/web

docker build -t td-react-image-dev

docker run -d --name td-react-dev --network caddy-dev-net td-react-image-dev
```

해당 서비스는 외부로 노출하지 않고 **반드시 Caddy를 통해 프록싱되어 접근**되어야 합니다. 즉, **Docker Network**를 적절히 구성하여 Caddy를 통해서만 접근 가능하도록 설정하여야 합니다.

위의 설정으로 띄워지는 컨테이너는 다음과 같습니다.

- td-react-dev 