11. 포팅 매뉴얼

프로젝트 개요

- 프로젝트 설명

Directory 구조

```
/S12P1B106
├── AI
│   ├── __init__.py
│   └── server
├── Backend
│   ├── Dockerfile.dev
│   ├── __pycache__
│   ├── ai
│   ├── alembic.ini
│   ├── auth
│   ├── data
```

```
│   ├── database.py
│   ├── discussion
│   ├── document_path
│   ├── main.py
│   ├── migrate.py
│   ├── migrations
│   ├── models.py
│   ├── novel
│   ├── requirements.txt
│   ├── scripts
│   ├── static
│   ├── user
│   └── utils
├── Docs
│   └── 프로젝트 기획서.docx
├── Frontend
│   ├── Dockerfile.dev
│   ├── README.md
│   ├── eslint.config.js
│   ├── index.html
│   ├── node_modules
│   ├── package-lock.json
│   ├── package.json
│   ├── public
│   ├── src
│   ├── tailwind.config.js
│   └── vite.config.js
├── README.md
├── app
│   └── controllers
├── certbot
│   ├── conf
│   └── www
├── docker-compose.override.yml
├── docker-compose.yml
├── generated_image.png
├── nginx
│   ├── conf.d
│   ├── letsencrypt
│   ├── ssl
│   └── www
├── package-lock.json
├── test_ci.sh
└── webRTC
    ├── README.md
    ├── backend
    └── frontend
```

Tech Stack Version

Framework / Library	Version	Description
docker	27.4.0	

docker-compsoe	2.31.0	
nginx	1.27.4	Web server
FastAPI	0.115.8	Python 웹 프레임워크
MariaDB	10.3.23	오픈소스 관계형 데이터베이스
Redis	6.0.2	인메모리 데이터 구조 저장소
Node.js	22.13.1	
NPM	10.9.2	
react	^19.0.0	UI 구축을 위한 JavaScript 라이브러리
react-dom	^19.0.0	React의 DOM 렌더링 패키지
react-router-dom	^7.1.5	React 애플리케이션의 라우팅 관리
@mui/material	^6.4.3	Material Design 기반 UI 컴포넌트 라이브러리
@mui/icons-material	^6.4.3	Material Design 아이콘 세트
@mui/joy	^5.0.0-beta.51	MUI의 새로운 컴포넌트 라이브러리
openvidu-browser	^2.31.0	WebRTC 기반 화상 통화 클라이언트 라이브러리
openvidu-node-client	^2.31.0	OpenVidu 서버와의 통신을 위한 Node.js 클라이언트
recordrtc	^5.6.2	웹 기반 미디어 녹화 라이브러리
wavefile	^11.0.0	WAV 오디오 파일 처리 라이브러리
axios	^1.7.9	HTTP 클라이언트 라이브러리
dayjs	^1.11.13	경량화된 날짜 처리 라이브러리

EC2 Ports

Skill	**Port**
FastAPI	8000
MariaDB	3306
Redis	6379
Nginx	81/443
OpenVidu	80/40000

환경 변수

Aa 변수명	설명	분류	값	비고
ACCESS_TOKEN_EXPIRE_MINUTES	JWT access token 만료 시간(분)	backend	30	
ALGORITHM	JWT 토큰 생성 알고리즘	backend	HS256	
DB_HOST	데이터베이스 호스트 주소	backend		
DB_MAX_OVERFLOW	연결 풀에서 허용하는 최대 연결 초과 개수	backend	10	
DB_NAME	데이터베이스 이름	backend	s12p11b106	

Aa 변수명	설명	분류	값	비고
DB_PASSWD	데이터베이스 비밀번호	backend		
DB_POOL_PRE_PING	데이터베이스 연결 유효성 검사 여부 확인	backend	true	
DB_POOL_RECYCLE	데이터베이스 연결 재활용 시간(초)	backend	3600	
DB_POOL_SIZE	데이터베이스 연결 풀 초기 연결 개수	backend	5	
DB_PORT	데이터베이스 포트 번호	backend	3306	
DB_USER	데이터베이스 DB 이름	backend	S12P11B106@stg-yswa-kr-practice-db-master.mariadb.database.azure.com	제공된 DB name
DEPLOY_SSH_PRIVATE_KEY	배포 서버에 접근하기 위한 키	SSH		
DOCKER_PASSWORD	dockerhub 비밀번호(토큰)	docker		
DOCKER_USERNAME	dockerhub username	docker		
ENVIRONMENT	백엔드 개발 환경 설정(JWT, 소셜 로그인 쿠키 설정을 위함)	backend	production	
GEMINI_API_KEY	gemini api를 사용하기 위한 key	AI		
GOOGLE_CLIENT_ID	구글 소셜 로그인 클라이언트 아이디	backend		Google Console에서 OAuth 2.0 등록 필요
GOOGLE_CLIENT_SECRET	구글 소셜 로그인 클라이언트 시크릿	backend		Google Console에서 OAuth 2.0 등록 필요
GOOGLE_TOKEN_URL	구글 소셜 로그인 토큰 발급 url	backend	https://oauth2.googleapis.com/token	Google Console에서 OAuth 2.0 등록 필요

Aa 변수명	설명	분류	값	비고
GOOGLE_USERINFO_URL	구글 소셜로그인 사용자 정보 반환 url	backend	https://www.googleapis.com/oauth2/v2/userinfo	Google Console에서 OAuth 2.0 등록 필요
JUPYTER_URL	AI 이미지 생성 URL	backend		
MAIL_FROM	fast-email(이메일 인증) 전송 유저 이름	backend		fast-email 라이브러리 설치 필요
MAIL_PASSWORD	fast-email(이메일 인증) 유저 비밀번호	backend		fast-email 라이브러리 설치 필요
MAIL_PORT	fast-email(이메일 인증) 포트번호	backend		fast-email 라이브러리 설치 필요
MAIL_SERVER	fast-email(이메일 인증) 서버 주소	backend	smtp.gmail.com	fast-email 라이브러리 설치 필요
MAIL_USERNAME	fast-email(이메일 인증) 유저 아이디	backend		fast-email 라이브러리 설치 필요
REDIS_HOST	REDIS 호스트	backend	redis	
REDIS_PORT	REDIS 포트번호	backend	6379	
REFRESH_TOKEN_EXPIRE_DAYS	refresh token 만료시간(일)	backend	7	
SECRET_KEY	JWT 생성, 검증 시 사용되는 시크릿 키	backend		
SSH_HOST	연결할 서버 주소	SSH		
TWILIO_ACCOUNT_SID	전화번호 인증 서비스 계정 SID	backend		Twilio API 등록 필요
TWILIO_AUTH_TOKEN	전화번호 인증 서비스 auth Token	backend		Twilio API 등록 필요
TWILIO_VERIFY_SID	전화번호 인증 서비스 인증 SID	backend		Twilio API 등록 필요
VITE_BACKEND_IP	frontend에서 호출하는	frontend	momoso106.duckdns.org	로컬 서버의 경우

Aa 변수명	설명	분류	값	비고
	API ip			localhost
VITE_BACKEND_PORT	frontend에서 호출하는 API port	frontend	''(빈 값)	로컬 서버의 경우 8000
VITE_BACKEND_PROTOCOL	frontend에서 호출하는 API protocol	frontend	https	로컬 서버의 경우 http
VITE_OPENVIDU_IP	frontend에서 openvidu를 호출하는 ip	frontend	43.202.64.156	배포 공용 ip에 따라 바뀌어야 합니다.
VITE_OPENVIDU_PORT	frontend에서 openvidu를 호출하는 port	frontend	40000	
VITE_OPENVIDU_PROTOCOL	frontend에서 openvidu를 호출하는 protocol	frontend	https	
VITE_OPENVIDU_SERVER_SECRET	frontend에서 openvidu를 호출하는 secrete key	frontend	root1234	
IMGUR_CLIENT_ID	이미지 드라이브(IMGUR) 클라이언트 ID	backend		
IMGUR_CLIENT_SECRETE	이미지 드라이브(IMGUR) 클라이언트 SECRETE	backend		
FRONTEND_REDIRECT_URI	구글 소셜 로그인 이후 redirect되는 URI	backend	https://momoso106.duckdns.org	
REDIRECT_URI	구글 소셜 로그인 콜백 리다이렉션 URL	backend	https://momoso106.duckdns.org/api/v1/oauth/google/callback	Google Console에 등록 필요

- https://momoso106.duckdns.org는 제공받은 EC2 서버의 공용 ip에 대한 domain name입니다.
  - 만약 ip가 변경된다면 새로운 domain name이 들어가야 합니다.

외부 API

- GeminiAPI

개발 환경

docker

`Dockerfile.dev`

- Backend/Dockerfile.dev

```
# Frontend/Dockerfile.dev
FROM python:3.13

WORKDIR /app

COPY requirements.txt /app

RUN pip install --no-cache-dir -r requirements.txt

COPY . /app

# 마이그레이션 스크립트에 실행 권한 부여
RUN chmod +x /app/migrate.py

WORKDIR /app

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000", "--reload", "--proxy-headers", "--forwarded-allow-i
```

  - Backend(FastAPI) 개발 환경의 docker 이미지를 만드는 Dockerfile이며 자세한 내용은 다음과 같습니다.
    1. OpenVidu의 구동을 위해 python:3.13 버전을 사용
    2. `requirements.txt`로 필요한 패키지를 설치
    3. 변경 사항에 대한 migration 진행
    4. uvicorn으로 FastAPI 서버 실행
       a. localhost:8000
       b. `--reload`로 개발 중 변경 사항이 바로 반영되도록 설정
  - Build

```
# project root에서 실행
docker build -t kwon0528/b106-backend:develop -f Backend/Dockerfile.dev ./Backend
```

- Frontend/Dockerfile.dev

```
# frontend/Dockerfile.dev
FROM node:22.13.1

# 작업 디렉토리 설정
WORKDIR /app

# package.json과 package-lock.json을 먼저 복사
COPY package*.json ./

# 의존성 설치
```

```
RUN npm install

# 소스 코드 복사
COPY . /app

WORKDIR /app

# 컨테이너가 시작될 때 실행될 명령어 설정
CMD ["npm", "run", "dev"]
```

- Frontend(React) 개발 환경의 docker 이미지를 만드는 Dockerfile이며 자세한 내용은 다음과 같습니다.
  1. LTS 버전인 node:22.13.1 사용
  2. package 관련 파일을 기반으로 필요한 모듈 설치
  3. 소스코드를 복사한 후 `npm run dev`로 개발 서버 구동
- Build

```
# project root에서 실행
docker build -t kwon0528/b106-frontend:develop -f Frontend/Dockerfile.dev ./Frontend
```

docker-compose.override

```
services:

  backend:
    image: kwon0528/b106-backend:develop
    container_name: dev-backend
    environment:
      DATABASE_URL: mysql+pymysql://$DB_USER:$DB_PASSWD@$DB_HOST:$DB_PORT/$DB_NAME?charset=utf8
    ports:
      - "8000:8000"
    volumes:
      - ./Backend:/app  # 로컬 파일 시스템을 컨테이너에 마운트
    networks:
      - app_network
    restart: unless-stopped

  frontend:
    image: kwon0528/b106-frontend:develop
    container_name: dev-frontend
    environment:
      - NODE_ENV=development
    ports:
      - "5173:5173"
    volumes:
      - ./Frontend:/app  # 로컬 파일 시스템을 컨테이너에 마운트
      - /app/node_modules
    networks:
      - app_network

  redis:
    image: 'redis:latest'
```

```
    ports:
      - 6379:6379
    volumes:
      - ./redis/data:/data
      - ./redis/conf/redis.conf:/usr/local/conf/redis.conf
    restart: always
    command: redis-server /usr/local/conf/redis.conf
    networks:
      - app_network

volumes:
  db_data:

networks:
  app_network:
    driver: bridge
```

- 위의 Dockerfile을 기반으로 개발 환경을 구성하는 docker-compose입니다
- Dockerfile을 기반으로 미리 만들어진 image를 통해 관리됩니다.
- 개발 중 변경 사항을 바로 반영할 수 있도록 host와 volume이 mount 되어 있습니다.
- 실행

```
# project root에서 실행
docker compose up
```

배포(운영) 환경

docker

`Dockerfile.prod`

- Backend/Dockerfile.prod

```
# Frontend/Dockerfile.dev
FROM python:3.13

WORKDIR /app

COPY requirements.txt /app

RUN pip install --no-cache-dir -r requirements.txt

COPY . /app

# 마이그레이션 스크립트에 실행 권한 부여
RUN chmod +x /app/migrate.py

WORKDIR /app

CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000", "--reload", "--proxy-headers", "--forwarded-allow-i
```

- Backend(FastAPI) 배포 환경의 docker 이미지를 만드는 Dockerfile이며 자세한 내용은 다음과 같습니다.
  1. OpenVidu의 구동을 위해 python:3.13 버전을 사용
  2. `requirements.txt`로 필요한 패키지를 설치
  3. 변경 사항에 대한 migration 진행
  4. uvicorn으로 FastAPI 서버 실행
     a. 배포 ip의 8000 포트 사용
- Build

```
# project root에서 실행
# release 배포
docker build -t kwon0528/b106-backend:release -f Backend/Dockerfile.prod ./Backend
# master 배포
docker build -t kwon0528/b106-backend:latest -f Backend/Dockerfile.prod ./Backend
```

- Frontend/Dockerfile.dev

```
# Multi-stage Build

# Build
FROM node:22.13.1 AS build

# 작업 디렉토리 설정
WORKDIR /app

# package.json과 package-lock.json을 먼저 복사
COPY package*.json ./

# # 의존성 설치
RUN npm install

# 나머지 애플리케이션 소스 코드 복사
COPY . .

# 컨테이너가 시작될 때 실행될 명령어 설정
# CMD ["tail", "-f", "/dev/null"]
RUN npm run build

RUN ls -la /app/dist || (echo "❌ build/ 폴더 없음! 빌드 실패" && exit 1)

# 최종 컨테이너에서 `build/` 폴더만 유지
FROM alpine AS final

WORKDIR /app
COPY --from=build /app/dist ./dist

RUN mkdir frontend_build
RUN ls -la /app/dist || (echo "❌ dist/ 폴더 없음! COPY 실패" && exit 1)

CMD ["cp", "-r", "/app/dist", "/app/frontend_build"]
```

- Frontend(React) 개발 환경의 docker 이미지를 만드는 Dockerfile이며 자세한 내용은 다음과 같습니다.

1. Build
    a. LTS 버전인 node:22.13.1 사용
    b. package 관련 파일을 기반으로 필요한 모듈 설치
    c. 소스코드를 복사한 후 `npm run build`로 배포를 위한 정적 파일 생성
2. Final
    a. alpine 기반의 최종 컨테이너를 열어서 정적 파일만 workspace로 복사
    b. entrypoint로 nginx와 공유하는 directory에 최종 복사
        i. 이렇게 해주지 않으면 volume이 mount될 때 host의 directory 상태가 덮어씌워집니다.

- Build

```
# project root에서 실행
# release 배포
docker build -t kwon0528/b106-frontend:release -f Frontend/Dockerfile.dev ./Frontend
# master 배포
docker build -t kwon0528/b106-frontend:release -f Frontend/Dockerfile.dev ./Frontend
```

docker-compose

```
services:
  backend:
    image: kwon0528/b106-backend:release
    container_name: prod-backend
    environment:
      DATABASE_URL: "mysql+pymysql://$DB_USER:$DB_PASSWD@$DB_HOST:$DB_PORT/$DB_NAME?charset=utf8"
    ports:
      - "8000:8000"
    networks:
      - app_network
    restart: unless-stopped

  frontend:
    image: kwon0528/b106-frontend:release
    container_name: prod-frontend
    volumes:
      - ./frontend_build:/app/frontend_build
    ports:
      - "5173:5173"
    networks:
      - app_network

  nginx:
    image: nginx:latest
    container_name: nginx
    restart: unless-stopped
    volumes:
      - ./frontend_build:/app/frontend_build
      - ./nginx/conf.d:/etc/nginx/conf.d
      - ./certbot/www:/var/www/certbot
      - ./certbot/conf:/etc/letsencrypt
    ports:
```

```
      - "81:80"  # 변경된 포트
      - "443:443"
    depends_on:
      - backend
      - frontend
    networks:
      - app_network

  redis:
    image: 'redis:latest'
    ports:
      - 6379:6379
    volumes:
      - ./redis/data:/data
      - ./redis/conf/redis.conf:/usr/local/conf/redis.conf
    restart: always
    command: redis-server /usr/local/conf/redis.conf
    networks:
      - app_network

volumes:
  db_data:

networks:
  app_network:
    driver: bridge
```

- 배포 환경에 맞는 이미지를 기반으로 환경을 실행합니다.
- frontend는 build 파일을 host에 전달하여 nginx가 정적 파일을 기반으로 서비스할 수 있도록 해줍니다.
- nginx는 volume의 `- ./frontend_build:/app/frontend_build` 를 통해 배포를 위한 정적 파일을 받아볼 수 있습니다.

WAS(Nginx)

```
# User-Agent 기반 Scraper 차단
map $http_user_agent $bad_bot {
    default 0;
    "~*curl" 1;
    "~*wget" 1;
    "~*python" 1;
    "~*httpclient" 1;
    "~*java" 1;
    "~*scrapy" 1;
    "~*postman" 1;
}

# Nginx Rate Limiting
limit_req_zone $binary_remote_addr zone=one:10m rate=5r/s;

server {
    listen 81;
    server_name momoso106.duckdns.org;
```

```
    location /.well-known/acme-challenge/ {
        root /var/www/certbot;
        allow all;
    }

    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name momoso106.duckdns.org;

    ssl_certificate /etc/letsencrypt/live/momoso106.duckdns.org/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/momoso106.duckdns.org/privkey.pem;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;
    ssl_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH";
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 10m;

    root /app/frontend_build/dist;
    index index.html;

    location / {
        try_files $uri /index.html;
    }

    location ~* \.(?:ico|css|js|gif|jpe?g|png|woff2?|eot|ttf|otf|svg|map|json)$ {
        expires max;
        log_not_found off;
    }

    location ~ /\.(?!well-known) {
        deny all;
        access_log off;
        log_not_found off;
    }

    # ✅ 🚀 **API 요청을 백엔드(FastAPI, Django 등)로 프록시 설정**
    location /api/ {
        proxy_pass http://backend:8000/api/;  # ✅ 백엔드 컨테이너 주소 (FastAPI, Django 등)
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

- OpenVidu가 80 포트를 점유하고 있어 81번 포트로 우회하여 http 통신을 합니다.
  - 인증서 생성 이외에는 모두 https를 사용하도록 합니다.
- certbot 기반으로 만들어진 SSL 인증서로 https 통신을 합니다.
  - DNS는 momoso106.duckdns.org입니다.

- 기본 endpoint(’/’)는 frontend 페이지로 이동하도록 합니다.
- /api/ 경로는 backend가 받을 수 있도록 합니다.

CI/CD pipeline

`.gitlab-ci.yml`

```
stages:
  - build
  - deploy

variables:
  IMAGE_BACKEND: kwon0528/backend
  IMAGE_FRONTEND: kwon0528/frontend
  TAG: $CI_COMMIT_REF_NAME

before_script:
  - docker --version
  - docker-compose --version

  - docker info
  - echo "$DOCKER_PASSWORD" | docker login -u "$DOCKER_USERNAME" --password-stdin

  - mkdir -p ~/.ssh
  - echo "🔑 Set ENV"

  - echo "ENV_VARIABLE=$ENV_VARIABLE" >> .env
  - ...

build_backend:
  stage: build
  tags:
    - backend-runner
  script:
    - cd Backend
    - |
      if [[ "$TAG" == "develop" ]]; then
        docker build -t $IMAGE_BACKEND:$TAG -f Dockerfile.dev .
      else
        docker build -t $IMAGE_BACKEND:$TAG -f Dockerfile.prod .
      fi
    - docker push $IMAGE_BACKEND:$TAG
  only:
    - develop
    - release
    - master

build_frontend:
  stage: build
  tags:
    - frontend-runner
  script:
```

```
    - cd Frontend
    - |
      if [[ "$TAG" == "develop" ]]; then
        docker build -t $IMAGE_FRONTEND:$TAG -f Dockerfile.dev .
      else
        docker build -t $IMAGE_FRONTEND:$TAG -f Dockerfile.prod .
      fi
    - docker push $IMAGE_FRONTEND:$TAG
  only:
    - develop
    - release
    - master

deploy:
  stage: deploy
  tags:
    - deploy
  before_script:
    - echo "$DEPLOY_SSH_PRIVATE_KEY" > id_rsa
    - chmod 600 id_rsa
    - ssh-keygen -p -m PEM -f id_rsa -N ""  # OpenSSH 호환 변환
    - mkdir -p ~/.ssh
    - ssh-keyscan -H $SSH_HOST > ~/.ssh/known_hosts  # SSH 호스트 키 등록
    - chmod 644 ~/.ssh/known_hosts
    - ssh -i id_rsa ubuntu@$SSH_HOST "echo SSH 연결 성공"

  script:
    - |
      ssh -i id_rsa ubuntu@$SSH_HOST << EOF
      set -e
      echo "Pull docker images..."
      docker pull $IMAGE_BACKEND:$TAG || { echo "Backend pull failed! Rolling back..."; exit 1; }
      docker pull $IMAGE_FRONTEND:$TAG || { echo "Frontend pull failed! Rolling back..."; exit 1; }

      echo "Checking if docker-compose is running..."
      cd /home/ubuntu/S12P11B106
      if [ "$(docker ps -q)" ]; then
        echo "Stopping running services..."
        docker-compose down
      else
        echo "No running containers found."
      fi

      echo "Starting new docker-compose services..."
      docker-compose -f docker-compose.yml up -d
      EOF

  only:
    - master
    - release
```

- 각 job들은 충돌을 피하기 위하여 지정된 gitlab runner를 사용합니다. (`tags` 로 확인 가능)

before_script

- dockerhub에 로그인합니다.
- `.env`로 관리하던 환경 변수들을 설정해줍니다. (CI/CD Variables 사용)

build stage

- 현재 branch에 기반하여 알맞은 Dockerfile로 image를 build하고 dockerhub에 push합니다.
- 이 stage는 develop, release, master에서 수행됩니다.

deploy stage

- 지급 받은 pem을 기반으로 서버와 연결합니다.
- 만들어진 image를 pull 하고 docker-compose로 서비스를 시작합니다.
  - 만약 이미 시작되어 있는 서비스가 있다면 중단 후 재시작합니다.
- 이 stage는 release, master에서 수행됩니다.

CI/CD test

ci-test.sh

```
#!/bin/bash

set -e  # 에러 발생 시 즉시 종료

echo "🚀 Starting local GitLab CI/CD pipeline test..."

export $(grep -v '^#' .env | xargs)

# 1️⃣ GitLab Runner 환경과 동일하게 Docker 컨테이너 내부에서 실행
docker run --rm \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v "$(pwd)":/workspace \
  -w /workspace \
  --env DOCKER_USERNAME="$DOCKER_USERNAME" \
  --env DOCKER_PASSWORD="$DOCKER_PASSWORD" \
  --env DEPLOY_SSH_PRIVATE_KEY="$DEPLOY_SSH_PRIVATE_KEY" \
  docker:latest sh -c '
    set -e
    apk add --no-cache docker-compose


    # 2️⃣ Docker 로그인
    echo "🔑 Logging in to Docker..."
    echo "$DOCKER_PASSWORD" | docker login -u "$DOCKER_USERNAME" --password-stdin

    # 3️⃣ Backend 빌드 및 푸시
    echo "🏗  Building Backend..."
    docker build -t kwon0528/b106-backend:release -f Backend/Dockerfile.prod ./Backend
    docker push kwon0528/b106-backend:release

    # 4️⃣ Frontend 빌드 및 푸시
    echo "🎨 Building Frontend..."
    docker build -t kwon0528/b106-frontend:release -f Frontend/Dockerfile.prod ./Frontend
```

```
    docker push kwon0528/b106-frontend:release


    # 1️⃣ Docker 컨테이너 실행
    echo "🧪 Running Tests..."
    BACKEND_CONTAINER_ID=$(docker run -d kwon0528/b106-backend:release)
    FRONTEND_CONTAINER_ID=$(docker run -d kwon0528/b106-frontend:release)

    # 2️⃣ 컨테이너 로그 출력 (비동기 실행)
    echo "🔍 Checking Backend logs..."
    docker logs -f "$BACKEND_CONTAINER_ID" &
    BACKEND_LOG_PID=$!

    echo "🔍 Checking Frontend logs..."
    docker logs -f "$FRONTEND_CONTAINER_ID" &
    FRONTEND_LOG_PID=$!

    # 3️⃣ 컨테이너 상태 확인
    sleep 10  # 컨테이너가 충분히 실행될 시간을 줌
    BACKEND_STATUS=$(docker inspect -f '{{.State.Running}}' "$BACKEND_CONTAINER_ID")
    FRONTEND_STATUS=$(docker inspect -f '{{.State.Running}}' "$FRONTEND_CONTAINER_ID")

    if [[ "$BACKEND_STATUS" == "true" && "$FRONTEND_STATUS" == "true" ]]; then
        echo "✅ Both containers are running successfully!"
    else
        echo "❌ Error: One or both containers failed to start."
        docker ps -a  # 현재 실행 중인 컨테이너 목록 출력
        exit 1
    fi
'

echo "✅ Local CI/CD pipeline test completed successfully!"
```

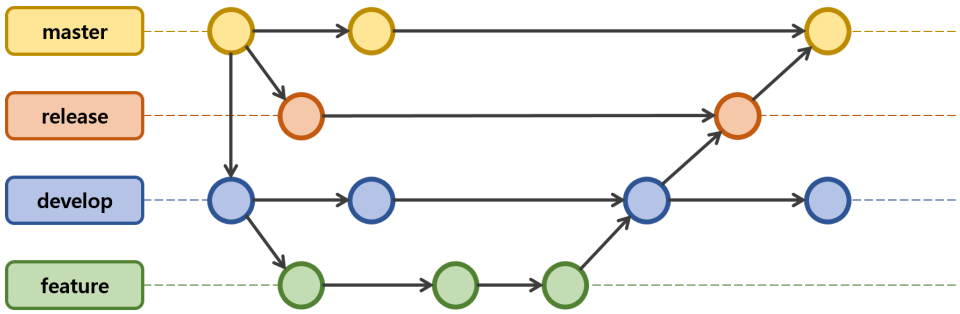
- 이 스크립트를 통해 `gitlab-ci.yml`을 local에서 어느 정도 구현해볼 수 있습니다. 이를 기반으로 최소한의 테스트를 진행하고 commit을 하게 됩니다.
- frontend / backend 코드를 기반으로 docker image를 build하고 dockerhub에 push하여 최신 버전의 image를 반영합니다.

Git 관리 전략

Git Flow

branch는 다음과 같이 관리됩니다.

Branch	Description
`master`	최종 배포가 이루어지는 기본 branch
`release`	배포 버전을 관리하는 branch
`develop`	개발 버전이 관리되는 branch
`feature/*`	기능 개발을 위한 branch



master

- 실제 최종 배포가 될 branch로 특별한 상황이 아니라면 직접 commit하지 않습니다.
- release에서 배포 test를 한 후 완료된 상태가 병합됩니다.

release

- 배포를 위한 변경 사항 이외에는 commit하지 않습니다.
  - 기능 개발, fix 관련 작업은 develop에서 진행합니다.
- develop에서 개발이 완료된 버전을 merge합니다.
  - 배포에 불필요한 개발 파일들은 제외합니다

```
#!/bin/sh

git merge --no-commit --no-ff develop

# 현재 브랜치 확인
CURRENT_BRANCH=$(git rev-parse --abbrev-ref HEAD)
echo CURRENT_BRANCH

# 병합 대상이 release 브랜치인 경우 특정 파일 삭제
if [ "$CURRENT_BRANCH" = "release" ]; then
    echo " 🔹 Merging into release branch - Removing unnecessary files..."

    FILES_TO_REMOVE=(
        ".env"
        "docker-compose.override.yml"
        "Dockerfile.dev"
        "*.log"
    )

    # 모든 경로에서 해당 파일을 검색하고 삭제
    for pattern in "${FILES_TO_REMOVE[@]}"; do
        # `git ls-files`를 사용하여 Git에 등록된 모든 파일 중에서 해당 패턴과 일치하는 파일 찾기
        git ls-files --cached --ignored --exclude="$pattern" | while read -r found_file; do
            # Git에서 삭제
            git rm -r "$found_file" 2>/dev/null && echo "✅ Removed from Git: $found_file" || echo "⚠️ Warning: $foun

            # 실제 파일 시스템에서 삭제
            rm -rf "$found_file" 2>/dev/null && echo "🗑️ Removed from disk: $found_file" || echo "⚠️ Warning: $found
        done
    done
```

```
    echo "✅ Selected files removed before merging into release branch."
fi
```

본 프로젝트에서는 merge 후 제외할 파일을 삭제하고 commit을 진행하는 스크립트를 기반으로 진행하였습니다.

develop

- 개발 환경 (docker 등) 이외의 사항은 일반적으로 commit하지 않습니다.
  - 기능 개발은 기능에 맞는 파생 branch를 만들고 작업을 진행합니다.
- feature branch에서 기능을 개발한 후 merge합니다.

feature

- 다양한 기능을 실제로 개발하는 branch입니다.
- branch 이름은 feature/<feature_name>으로 명명하고, develop에서 파생하여 만듧니다.
  - `git branch feature/<feature_name> develop`

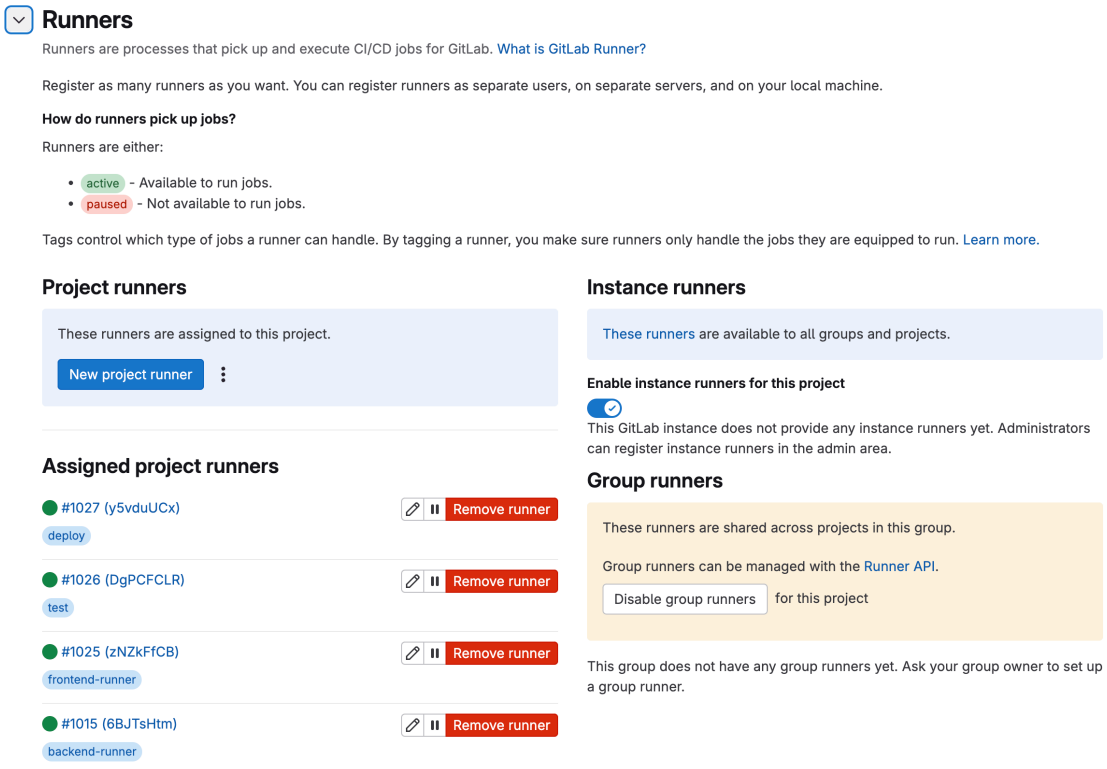
특이 사항

docker image

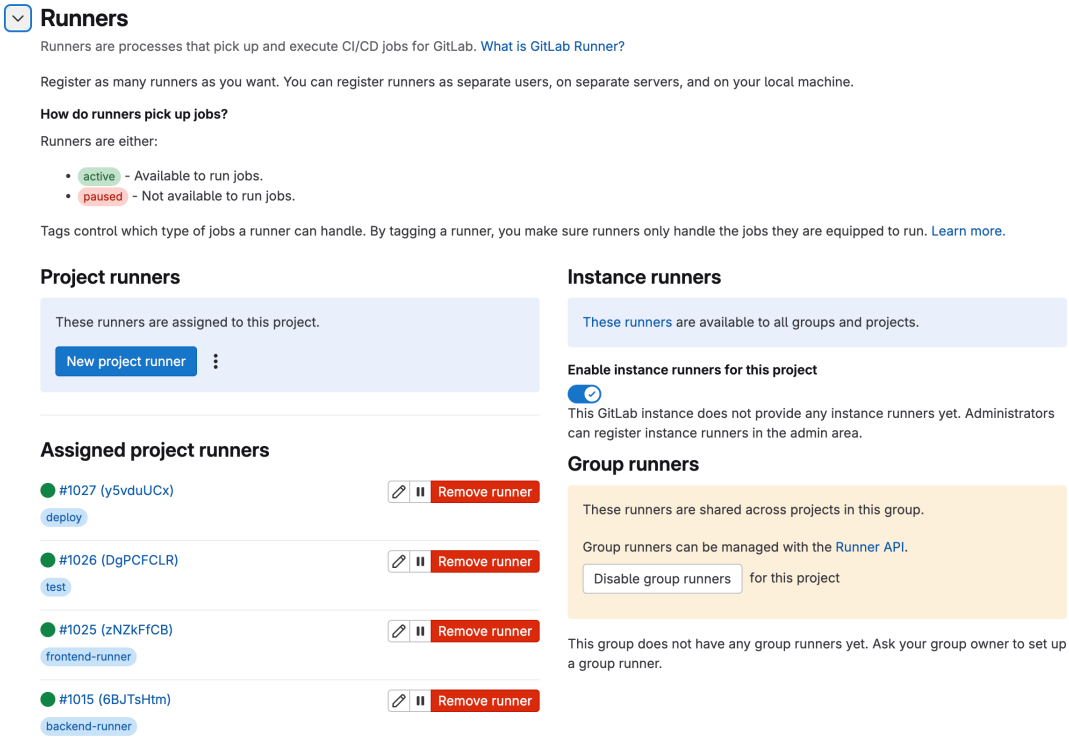
- 기본적으로 docker image는 인프라 담당 팀원(권기범)의 개인 docker hub 계정을 기반으로 pull / push가 이루어집니다.
  - 다른 dockerhub 계정을 사용한다면 `docker compose.yml` 과 `gitlab-ci.yml`을 수정하여 알맞은 dockerhub 계정으로 사용되도록 하여야 합니다.

배포

- 서버에 미리 master branch가 받아져 있어야 합니다.
- `gitlab-ci.yml` 의 `cd /home/ubuntu/S12P11B106` 부분은 프로젝트의 root 경로여야 합니다.
- 모든 환경 변수는 gitlab의 CI/CD variable에 입력되어 있어야 하며, .env도 `/home/ubuntu/S12P11B106`에 있어야 합니다.
- CI/CD pipeline이 작동하려면 gitlab runner가 동작하고 있어야 합니다.
  - 미리 gitlab runner를 설치하거나, gitlab runner의 docker conainer 위에서 아래 프로세스를 진행합니다.
    - gitlab runner는 gitlab의 **CI/CD Settings**에서 만들 수 있습니다.

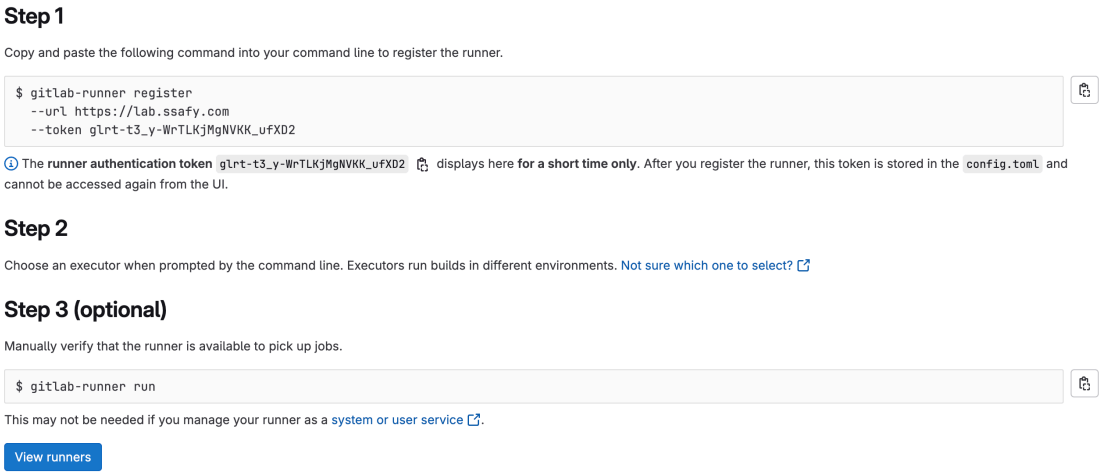


- New project runner를 클릭한 이후 runner의 tag를 설정하고 생성합니다.
  - backend-runner, frontend-runner, deploy 총 3가지의 tag로 runner를 생성합니다.



- 다음과 같은 절차를 따라 각 runner를 등록합니다.

- 모든 러너에 대해 Step 1을 실행하여 연결이 완료되었다면 `gitlab-runner run` 을 통해 모든 runner를 실행합니다.



Step 1

Copy and paste the following command into your command line to register the runner.

```
$ gitlab-runner register
  --url https://lab.ssafy.com
  --token glrt-t3_y-WrTLKjMgNVKK_ufXD2
```

The **runner authentication token** `glrt-t3_y-WrTLKjMgNVKK_ufXD2` displays here **for a short time only**. After you register the runner, this token is stored in the `config.toml` and cannot be accessed again from the UI.

Step 2

Choose an executor when prompted by the command line. Executors run builds in different environments. Not sure which one to select?

Step 3 (optional)

Manually verify that the runner is available to pick up jobs.

```
$ gitlab-runner run
```

This may not be needed if you manage your runner as a system or user service.

View runners

OpenVidu

- 서버를 띄운 ip 주소의 40000번 포트로 한 번 접근해주어야 토론 기능이 정상적으로 작동합니다,