포팅 메뉴얼

생성자	문영민
생성 일시	@2025년 5월 21일 오전 9:24
카테고리	개발 매뉴얼
최종 편집자:	문영민
최종 업데이트 시간	@2025년 5월 22일 오전 11:05

빌드 및 배포 방법

- 프로젝트 버전

	버전	port	비고
java	21.x		
Spring-boot	3.4.x	8081(lightreborn), 8082(dearie)	gradle 사용
next	15.3.1	3000(lightreborn), 3001(dearie)	
node.js	22		
Postgre	16	5432(lightreborn), 5433(dearie)	
redis		6379(lightreborn), 6380(dearie)	
react	^19.0.0		
zustand	^5.0.3		
jenkins		8080	
kafka		9092	
zookeeper		2181	
kafka-ui		8090	

서버 접속 & 설정

- EC2 서버 접속을 위해 .pem 파일을 사용하여 접속합니다.

```
ssh -i <pem키_경로> ubuntu@k12s309.p.ssafy.io
```

- EC2 서버에 도커를 다운로드 합니다.

```
# 1.
패키지 업데이트
sudo apt update && sudo apt upgrade -y

# 2. Docker 설치
sudo apt install docker.io -y

# 3. Docker 서비스 활성화
sudo systemctl start docker
sudo systemctl enable docker

# 4. 현재 사용자(ubuntu)를 Docker 그룹에 추가(재로그인 필요)
sudo usermod -aG docker $USER
```

✅ 재로그인(필수!)

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

💡AWS EC2에서 Permission Denied 오류가 발생하면?

```
sudo chmod 666 /var/run/docker.sock
```

이후 다시 실행.

- 컨테이너들을 쉽게 관리하기 위해 docker-compose를 설치합니다.

```
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docke
r-compose
sudo chmod +x /usr/local/bin/docker-compose
```

확인:

```
docker-compose --version
```

젠킨스 설치

- 젠킨스 설정 백업을 위해 마운트 할 볼륨의 디렉토리를 생성합니다.

```
cd /home/ubuntu && mkdir jenkins-data a
```

- 서버에 외부에서 접속할 포트를 오픈하고 상태를 확인합니다.

```
sudo ufw allow 8080 /tcp
sudo ufw reload
sudo ufw status
```

- docker 명령어로 jenkins container를 생성 및 구동합니다.

```
sudo docker run -d -p 8080:8080 -v /home/ubuntu/jenkins-data:/var/jenkins_home --name jenkins jenkins:jenkins:latest
```

- 환경 설정 변경을 위해 jenkins data 폴더로 이동합니다. update center에 필요한 CA 파일을 다운로드 합니다.

```
mkdir update-center-rootCAs
wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-center/rootCA/update-center.crt -0 ./update-center-rootCAs/update-center.crt
```

- 아래 명령어를 실행 후 `hudson.model.UpdateCenter.xml` 파일을 열어 `https://raw.githubusercontent.com/lework/jenkins-update-center/master/updates/tencent/update-center.json` 이 URL로 변경되었는지 확인하세요.

```
sudo sed -i 's#https://updates.jenkins.io/update-center.json#https://raw.githubusercontent.com/lework/jenkins-update-center/master/updates/tence
```

- jenkins 서비스를 구동합니다.

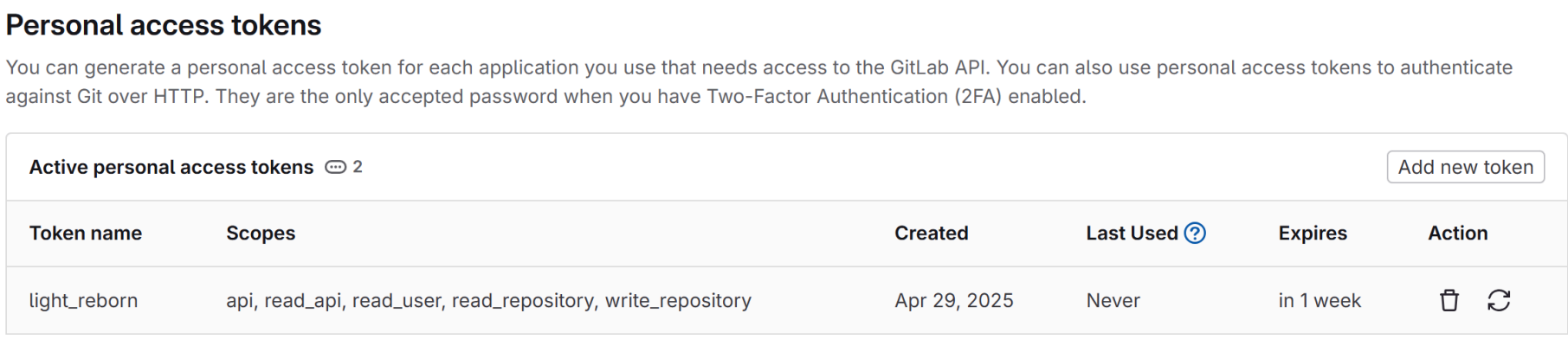
```
sudo docker restart jenkins
```

젠킨스 설정

- 원활한 gitlab 관리를 위해 플러그인을 설치합니다.
  - Generic Webhook Trigger Plugin
  - GitLab Branch Source Plugin
  - Pipeline Utility Steps

Crendential 설정

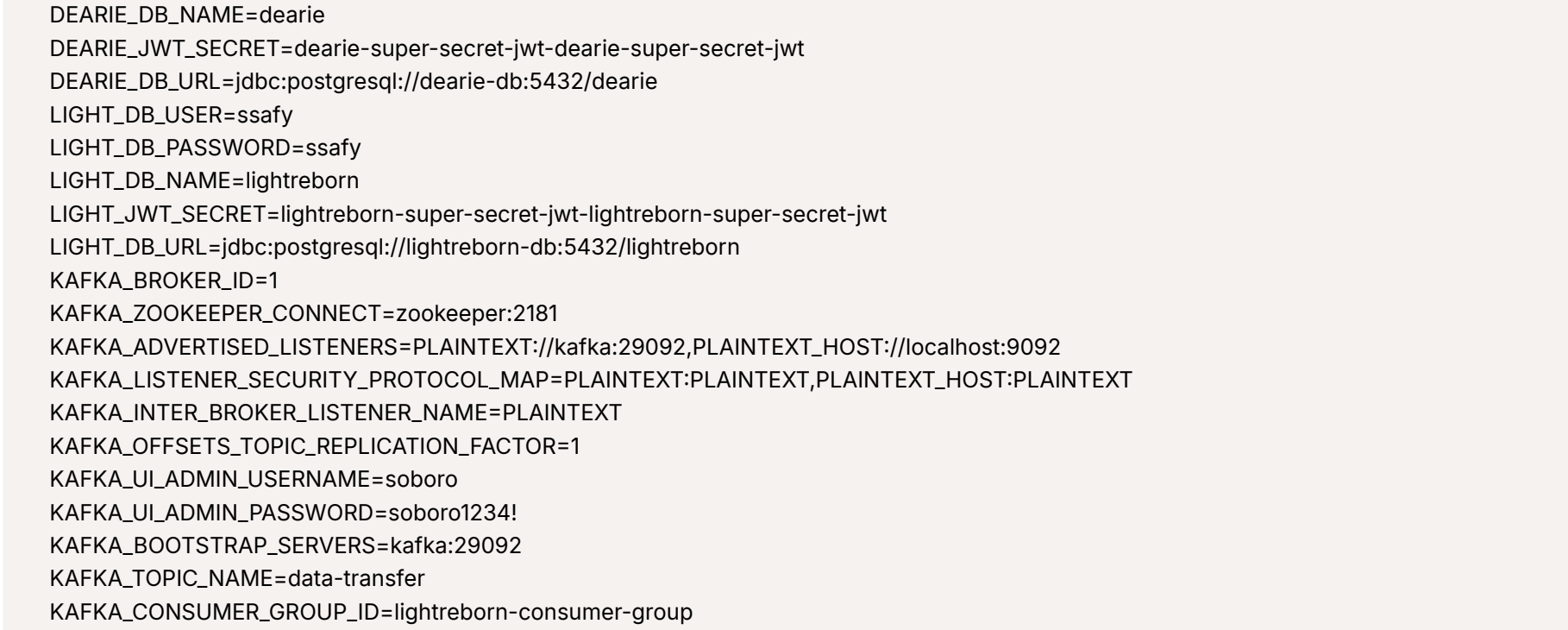
- GitLab의 프로필을 누른 후 Prereferences에서 Personal access tokens을 발급합니다.



생성 후 token을 복사합니다.

- 대시보드에서 Managa Jenkins를 누른 후 Credential에서 (global)에 발급받은 GitLab Personal Access token을 추가합니다.

  GitLab Personal Token: J_rc6btW39dwigwuoTdb

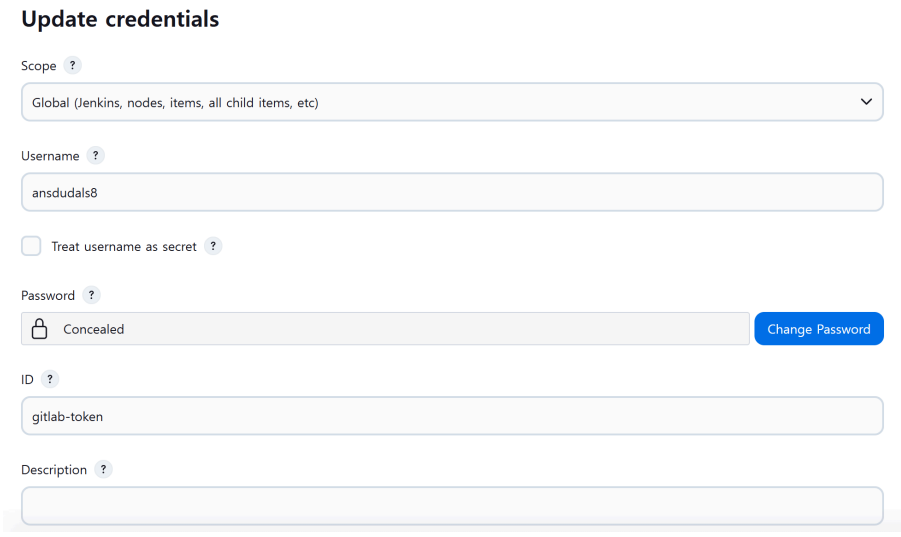


- DB 정보를 담고있는 파일을 Secret text로 등록합니다.

```
{
  spring.profiles.active=prod
  DEARIE_DB_USER=ssafy
  DEARIE_DB_PASSWORD=ssafy
  DEARIE_DB_NAME=dearie
  DEARIE_JWT_SECRET=dearie-super-secret-jwt-dearie-super-secret-jwt
  DEARIE_DB_URL=jdbc:postgresql://dearie-db:5432/dearie
  LIGHT_DB_USER=ssafy
  LIGHT_DB_PASSWORD=ssafy
  LIGHT_DB_NAME=lightreborn
  LIGHT_JWT_SECRET=lightreborn-super-secret-jwt-lightreborn-super-secret-jwt
  LIGHT_DB_URL=jdbc:postgresql://lightreborn-db:5432/lightreborn
  KAFKA_BROKER_ID=1
  KAFKA_ZOOKEEPER_CONNECT=zookeeper:2181
  KAFKA_ADVERTISED_LISTENERS=PLAINTEXT://kafka:29092,PLAINTEXT_HOST://localhost:9092
  KAFKA_LISTENER_SECURITY_PROTOCOL_MAP=PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
  KAFKA_INTER_BROKER_LISTENER_NAME=PLAINTEXT
  KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR=1
  KAFKA_UI_ADMIN_USERNAME=soboro
  KAFKA_UI_ADMIN_PASSWORD=soboro1234!
  KAFKA_BOOTSTRAP_SERVERS=kafka:29092
  KAFKA_TOPIC_NAME=data-transfer
  KAFKA_CONSUMER_GROUP_ID=lightreborn-consumer-group
  OPENAI_API_KEY=sk-proj-CEb6gKaCmXd8qnvDN7vpchD123BPza2TfZwnxZn4e9lu6xEWyNAuJXIVHhjWxzxy7-7AdrqzUXT3BlbkFJGjjWfuweq7Xr
  S3_ACCESS_KEY=AKIAWT5ZPCHYJLX2EY4B
  S3_SECRET_KEY=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
  S3_BUCKET_LIGHTREBORN=light-reborn-bucket
  S3_BUCKET_DEARIE=dearie-bucket
  NEXT_PUBLIC_NAVER_CLIENT_ID=q7q9fxdnmn
  NEXT_PUBLIC_MAPBOX_TOKEN=pk.eyJ1Ijoic29ib3JvMzA5IiwiYSI6ImNtYWx3bzRrNzBkZTEybnBrdjA3MnQ3cTgifQ.BFc5hAN8ITpIKF3E7jWwOQ
```

```
  NEXT_PUBLIC_MAPTILER_KEY=zDdBSm5yFRmFgBZ9bSIB
  KAKAO_REST_API_KEY=53ba64d73976935d1286d226089381ff
}
```

- gitlab 사용자 credential을 등록합니다.

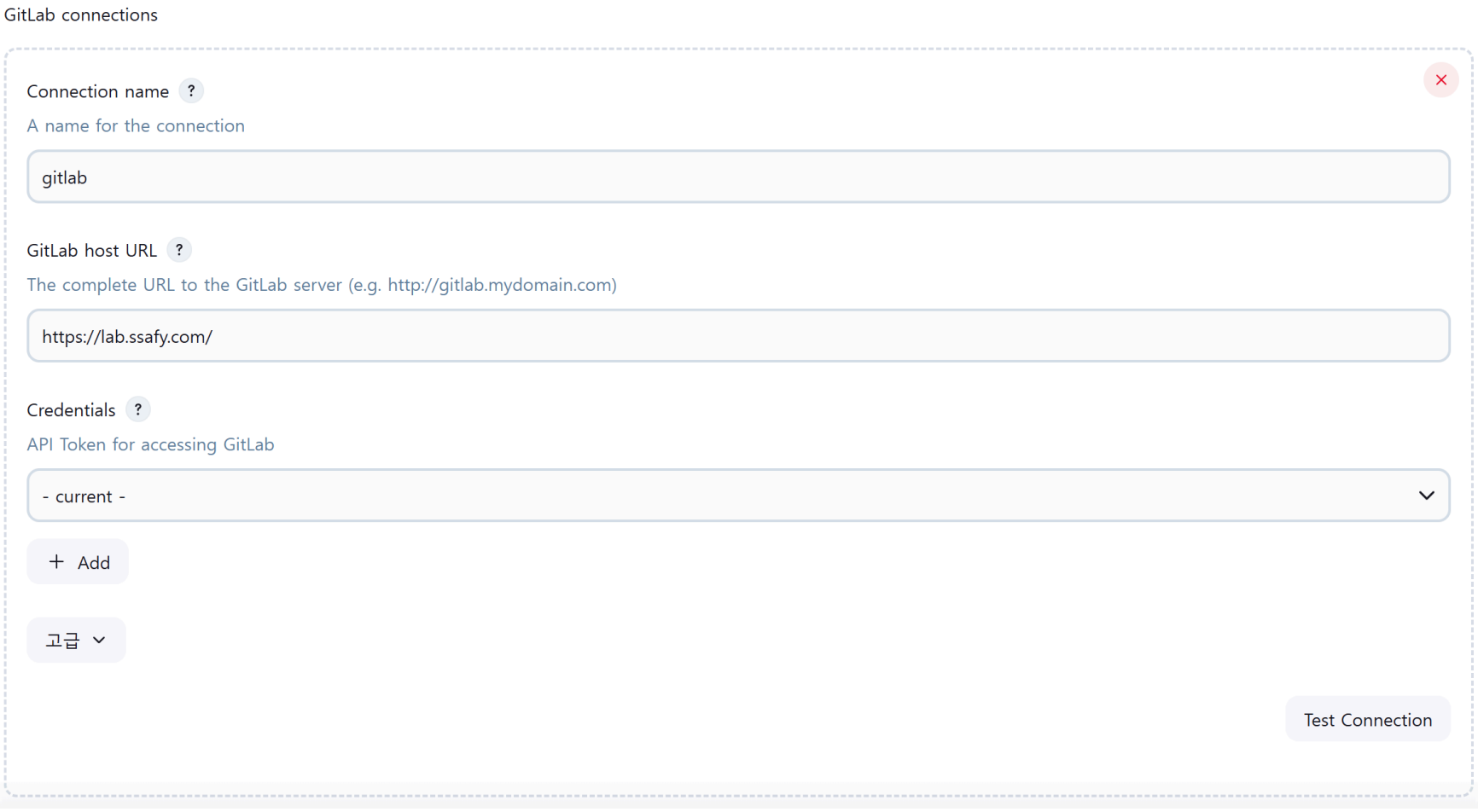


- EC2_IP와 MATTERMOST_WEBHOOK을 secret credential에 등록합니다.

```
EC2_IP:
MATTERMOST_WEBHOOK:
```

젠킨스 시스템 설정

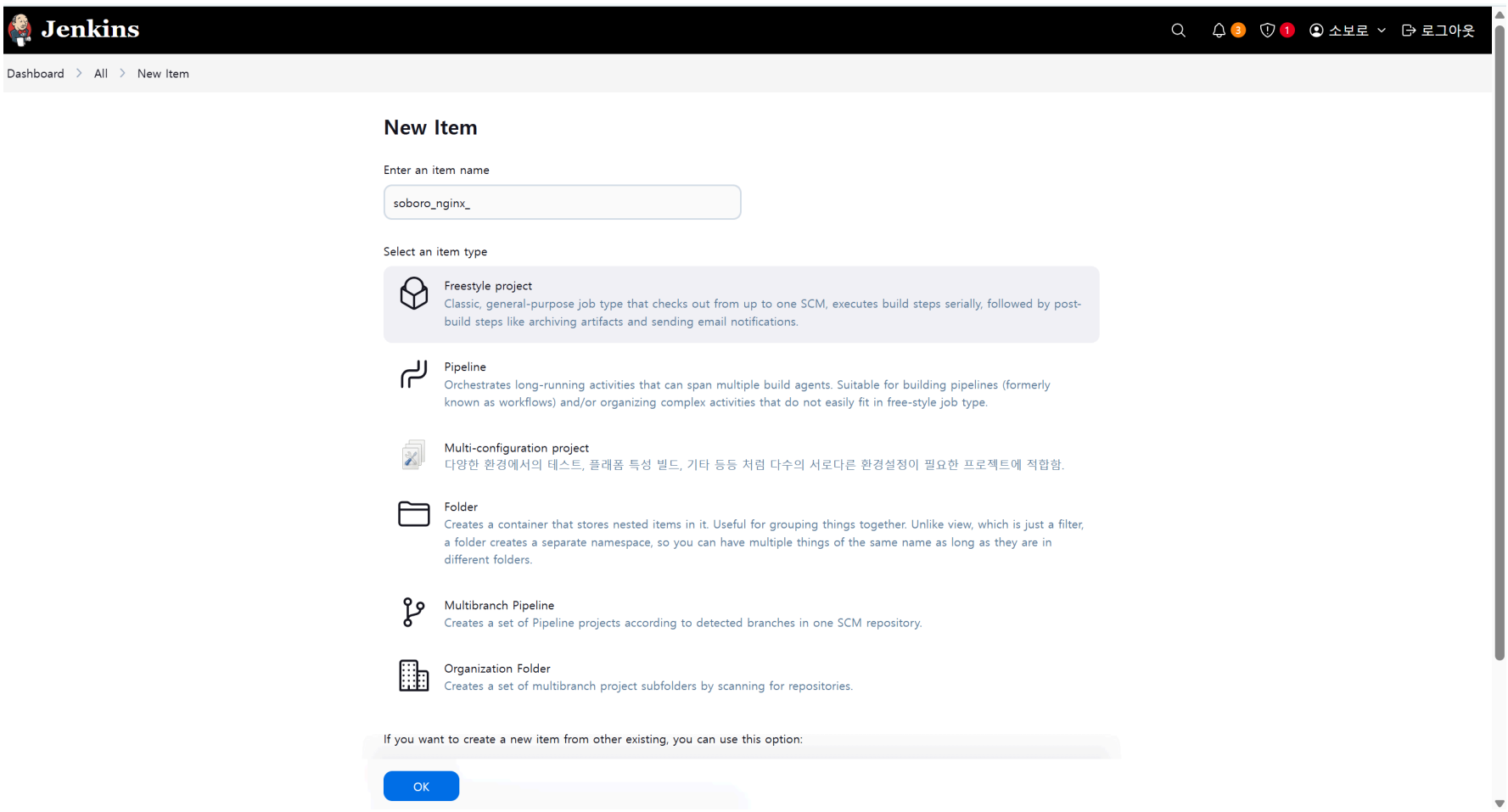
- System의 해당 탭에서 gitlab 설정 마칩니다.
  - GitLab



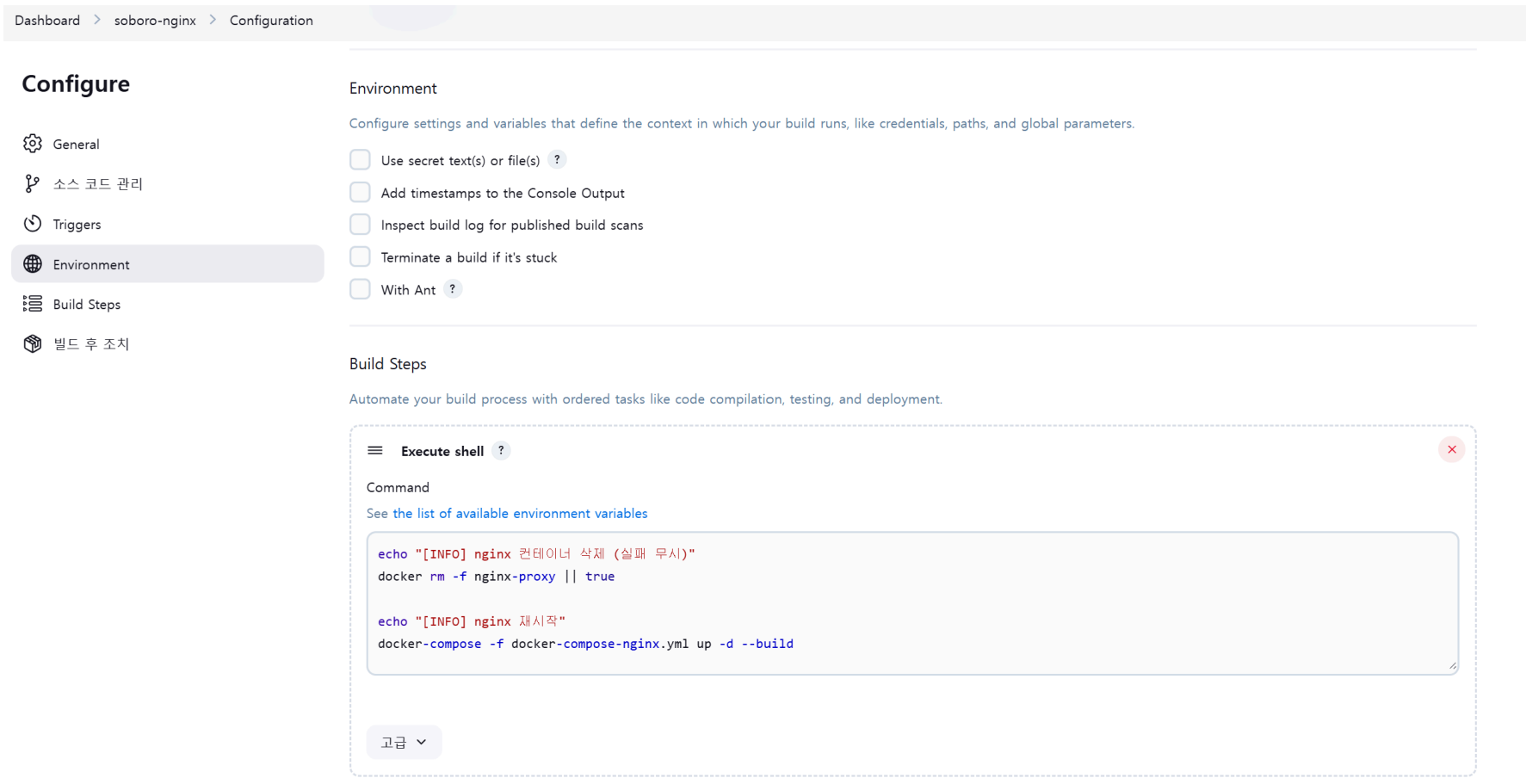
파이프라인 준비

- **Nginx**

1. 가용을 위한 파이프라인을 만듭니다.



2. build에 필요한 단계를 코드로 작성합니다.




```
echo "[INFO] nginx 컨테이너 삭제 (실패 무시)"
docker rm -f nginx-proxy || true

echo "[INFO] nginx 재시작"
docker-compose -f docker-compose-nginx.yml up -d --build
```

Job 설정에서 위의 코드를 붙여넣습니다.

3. EC2 서버에 빌드에 필요한 파일을 업로드 합니다.

```
ubuntu@ip-172-26-9-44:~/jenkins-data/workspace/soboro-nginx$ tree
.
├── certbot
│   ├── conf
│   │   ├── accounts
│   │   │   ├── acme-staging-v02.api.letsencrypt.org
│   │   │   │   └── directory
│   │   │   │       └── a2f2eefd271114fb0b96c1e95da2c667
│   │   │   │           ├── meta.json
│   │   │   │           ├── private_key.json
│   │   │   │           └── regr.json
│   │   │   └── acme-v02.api.letsencrypt.org
│   │   │       └── directory
│   │   │           ├── afecd72cd23335c22a3961e803d03605
│   │   │           │   ├── meta.json
│   │   │           │   ├── private_key.json
│   │   │           │   └── regr.json
│   │   │           └── ea8574c85b154ff336e632624abb7d62
│   │   │               ├── meta.json
│   │   │               ├── private_key.json
│   │   │               └── regr.json
│   │   ├── archive
│   │   │   └── k12s309.p.ssafy.io
│   │   │       ├── cert1.pem
│   │   │       ├── chain1.pem
│   │   │       ├── fullchain1.pem
│   │   │       └── privkey1.pem
│   │   ├── live
│   │   │   ├── README
│   │   │   └── k12s309.p.ssafy.io
│   │   │       ├── README
│   │   │       ├── cert.pem -> ../../archive/k12s309.p.ssafy.io/cert1.pem
│   │   │       ├── chain.pem -> ../../archive/k12s309.p.ssafy.io/chain1.pem
│   │   │       ├── fullchain.pem -> ../../archive/k12s309.p.ssafy.io/fullchain1.pem
│   │   │       └── privkey.pem -> ../../archive/k12s309.p.ssafy.io/privkey1.pem
│   │   ├── renewal
│   │   │   └── k12s309.p.ssafy.io.conf
│   │   └── renewal-hooks
│   │       ├── deploy
│   │       ├── post
│   │       └── pre
│   └── www
│       └── test.txt
├── default.conf
├── docker-compose-nginx.yml
```

파일 구조도

1. certbot 폴더
    a. www 폴더
    b. conf 폴더
2. `/jenkins-data/workspace/soboro-nginx` 폴더에 docker-compose-nginx.yml 파일을 업로드 합니다.

```
services:
  nginx:
    image: nginx:latest
    container_name: nginx-proxy
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - /home/ubuntu/jenkins-data/workspace/soboro-nginx/default.conf:/etc/nginx/conf.d/default.conf:ro
      - /home/ubuntu/jenkins-data/workspace/soboro-nginx/certbot/www:/var/www/certbot
      - /home/ubuntu/jenkins-data/workspace/soboro-nginx/certbot/conf:/etc/letsencrypt
      # - ./default.conf:/etc/nginx/conf.d/default.conf:ro
      # - ./certbot/www:/var/www/certbot
      # - ./certbot/conf:/etc/letsencrypt
    networks:
      - lightreborn-net
      - dearie-net


networks:
  lightreborn-net:
    external: true
  dearie-net:
    external: true
```

4. 빌드 버튼을 통해 nginx-proxy 컨테이너를 생성합니다.

- DBs(postgres-lightreborn, postgres-dearie)
    1. DB가용을 위한 파이프 라인을 만듭니다.
    2. 파이프라인 스크립트에 아래 코드를 붙여넣습니다.

```
pipeline {
    agent any

    stages {
        stage('Load .env File') {
            steps {
                withCredentials([string(credentialsId: 'soboro-dotenv', variable: 'DOTENV')]) {
```

```
            script {
                def envFilePath = "cicd/.env"

                // .env 파일 생성
                writeFile file: envFilePath, text: DOTENV

                // 생성 후 존재 여부 확인
                if (!fileExists(envFilePath)) {
                    error "❌ .env 파일이 ${envFilePath} 위치에 없습니다."
                }

                echo "✅ .env 파일을 Credentials로부터 생성 및 확인 완료"
            }
        }
    }
}
stage('Check Workspace Contents') {
    steps {
        echo "📂 Jenkins 현재 워크스페이스 구조 확인:"
        sh 'ls -al'
        sh 'ls -al cicd || echo cicd 디렉토리 없음'
    }
}

stage('Deploy DBs to EC2') {
    steps {
        withCredentials([
            sshUserPrivateKey(credentialsId: 'ec2-ssh-key', keyFileVariable: 'SSH_KEY', usernameVariable: 'SSH_USER'),
            string(credentialsId: 'EC2_IP', variable: 'EC2_IP')
        ]) {
            sh '''
            echo "📦 EC2에 db 디렉토리 생성..."
            ssh -o StrictHostKeyChecking=no -i $SSH_KEY $SSH_USER@$EC2_IP 'mkdir -p /home/ubuntu/db'

            echo "📤 .env 파일만 EC2로 복사 중..."
            scp -o StrictHostKeyChecking=no -i $SSH_KEY cicd/.env $SSH_USER@$EC2_IP:/home/ubuntu/db/

            echo "🐳 EC2에서 docker-compose 실행..."
            ssh -o StrictHostKeyChecking=no -i $SSH_KEY $SSH_USER@$EC2_IP '
                cd ~/db

                echo "🗑 기존 컨테이너 제거 시도"
                docker rm -f lightreborn-db dearie-db || true

                if [ -f .env ]; then
                    echo "🐳 docker compose 실행 시작"
                    docker compose -f docker-compose.db.yml --env-file .env up -d
                    echo "🧹 .env 파일 삭제"
                    rm -f .env
                else
                    echo "❌ .env 파일이 존재하지 않아 docker-compose 실행 중단"
                    exit 1
                fi
            '
            '''
        }
    }
}

    post {
        always {
            echo "🧹 로컬 .env 파일 삭제 중..."
            sh 'rm -f cicd/.env || true'
        }
    }
}
```

3. 깃 저장소의 root의 cicd폴더에 docker-compose.yml 파일을 업로드 합니다.

```
version: "3.9"

services:
  dearie-db:
    image: postgres:16
    container_name: dearie-db
    ports:
      - "5433:5432"
    env_file:
      - .env
    environment:
      - POSTGRES_USER=${DEARIE_DB_USER}
      - POSTGRES_PASSWORD=${DEARIE_DB_PASSWORD}
      - POSTGRES_DB=${DEARIE_DB_NAME}
    volumes:
      - postgres_data_dearie:/var/lib/postgresql/data
    networks:
      - dearie-net
    restart: unless-stopped

  lightreborn-db:
    image: postgres:16
    container_name: lightreborn-db
    ports:
      - "5432:5432"
    env_file:
      - .env
    environment:
      - POSTGRES_USER=${LIGHT_DB_USER}
      - POSTGRES_PASSWORD=${LIGHT_DB_PASSWORD}
      - POSTGRES_DB=${LIGHT_DB_NAME}
    volumes:
      - postgres_data_lightreborn:/var/lib/postgresql/data
    networks:
      - lightreborn-net
    restart: unless-stopped

volumes:
  postgres_data_dearie:
  postgres_data_lightreborn:

networks:
  dearie-net:
    external: true
  lightreborn-net:
    external: true
```

4. 파이프 라인 build 버튼을 통해 db 컨테이너들을 생성합니다.

Docker-compose, Dockerfile, JenkinsFile

깃랩의 웹훅으로 들어오는 이벤트를 통해 자동화를 진행하기 위해 해당 파일을 맞는 폴더로 이동시킵니다.

GitLab Repository

- Jenkinsfile
  - root폴더에 Jenkinsfile을 위치시킵니다.

```
def envProps
def buildSuccess = false

def generateEnvString = { keys ->
    keys.collect { key -> "${key}=${envProps[key]}" }.join('\n')
}

def generateWithEnvList = { keys ->
    keys.collect { key -> "${key}=${envProps[key]}" }
}

pipeline {
    agent any
```

```
parameters {
    choice(name: 'ENV', choices: ['develop', 'master'], description: 'Select deploy environment')
}

environment {
    MATTERMOST_WEBHOOK_ID = 'MATTERMOST_WEBHOOK'
}

stages {
    stage('Decide Environment') {
    // 0. 브랜치 기반 ENV 자동 설정
        steps {
            script {
                def branch = env.BRANCH_NAME ?: env.GIT_BRANCH ?: sh(script: "git rev-parse --abbrev-ref HEAD", returnStdout: true).trim()
                def selectedEnv = params.ENV?.trim()?.toLowerCase()
                def workspace = env.WORKSPACE.replaceFirst("^/var/jenkins_home", "/home/ubuntu/jenkins-data")

                if (!selectedEnv || !(selectedEnv in ['develop', 'master'])) {
                    selectedEnv = (branch == 'develop') ? 'develop' : 'master'
                    echo "🔄 ENV auto-detected as: ${selectedEnv}"
                } else {
                    echo "✅ ENV manually selected: ${selectedEnv}"
                }
                env.ENV = selectedEnv

                // env.CUSTOM_WORKSPACE = workspace
            }
        }
    }

// 1. 먼저 .env 파일부터 읽음
stage('Load .env File') {
    steps {
        withCredentials([string(credentialsId: 'soboro-dotenv', variable: 'DOTENV')]) {
            script {
                def envFilePath = "${env.WORKSPACE}/cicd/.env"

                def correctedContent = DOTENV.replaceAll(/([A-Z][A-Z0-9_]+)=/, '\n$1=').trim()

                writeFile file: envFilePath, text: correctedContent

                // 이제 제대로 파싱
                envProps = [:]
                correctedContent.readLines().each { line ->
                    if (line && line.contains('=') && !line.trim().startsWith('#')) {
                        def split = line.split('=', 2)
                            if (split.length == 2) {
                                envProps[split[0].trim()] = split[1].trim()
                            }
                        }

                echo "✅ .env 파일 읽기 완료: ${envProps.size()}개 프로퍼티"
                // echo "✅ 키 목록: ${envProps.keySet()}"
            }
        }
    }
}

// 2. generate env - backend
stage('Generate .env') {
    steps {
        script {
            def requiredVars = [
                'DEARIE_DB_URL', 'DEARIE_DB_USER', 'DEARIE_DB_PASSWORD', 'DEARIE_DB_NAME', 'DEARIE_JWT_SECRET',
                'LIGHT_DB_URL', 'LIGHT_DB_USER', 'LIGHT_DB_PASSWORD', 'LIGHT_DB_NAME', 'LIGHT_JWT_SECRET',
                'KAFKA_BOOTSTRAP_SERVERS', 'KAFKA_TOPIC_NAME', 'KAFKA_CONSUMER_GROUP_ID',
                'OPENAI_API_KEY', 'S3_ACCESS_KEY', 'S3_SECRET_KEY', 'S3_BUCKET_DEARIE', 'S3_BUCKET_LIGHTREBORN',
                'NEXT_PUBLIC_NAVER_CLIENT_ID', 'KAKAO_REST_API_KEY'
            ]

            requiredVars.each { var ->
```

```
                if (!envProps.containsKey(var)) {
                    error "❌ 필수 변수 ${var}가 envProps에 없습니다."
                }
            }

            def newEnvContent = generateEnvString(requiredVars) + '\nspring.profiles.active=prod'

            writeFile file: "${env.WORKSPACE}/cicd/.env", text: newEnvContent.trim()
            echo "✅ .env 재생성 완료"
        }
    }
}

// generate env - lightreborn-frontend
stage('Generate frontend .env.production') {
    steps {
        script {
            def frontendEnv = """
            NEXT_PUBLIC_NAVER_CLIENT_ID=${envProps.NEXT_PUBLIC_NAVER_CLIENT_ID}
            NEXT_PUBLIC_API_URL=/api/dashboard/
            """.stripIndent().trim()

            writeFile file: "${env.WORKSPACE}/lightreborn/frontend/.env.production", text: frontendEnv
            echo "✅ lightreborn frontend용 .env.production 생성 완료"
        }
    }
}

// generate env - dearie-frontend
stage('Generate frontend .env.dearie.production') {
    steps {
        script {
            def frontendEnv = """
            NEXT_PUBLIC_BASE_PATH=/dearie
            NEXT_PUBLIC_API_URL=/api/app/
            NEXT_PUBLIC_MAPBOX_TOKEN=${envProps.NEXT_PUBLIC_MAPBOX_TOKEN}
            NEXT_PUBLIC_MAPTILER_KEY=${envProps.NEXT_PUBLIC_MAPTILER_KEY}
            NEXT_PUBLIC_NAVER_CLIENT_ID=${envProps.NEXT_PUBLIC_NAVER_CLIENT_ID}
            KAKAO_REST_API_KEY=${envProps.KAKAO_REST_API_KEY}
            """.stripIndent().trim()

            writeFile file: "${env.WORKSPACE}/dearie/frontend/.env.dearie.production", text: frontendEnv
            echo "✅ dearie frontend용 .env.dearie.production 생성 완료"
        }
    }
}


// 3. 기존 컨테이너 정리
stage('Clean up Existing Containers') {
    steps {
        script {
            def composePath = "${env.WORKSPACE}/docker-compose.yml"
            def envPath = "${env.WORKSPACE}/cicd/.env"

            sh """
                echo \"🧹 docker-compose down (remove orphans)\"
                docker-compose --env-file ${envPath} -f ${composePath} down --remove-orphans || true

                docker rm -f dearie-backend lightreborn-backend dearie-frontend lightreborn-frontend || true
            """
        }
    }
}

// 4. 빌드 및 배포
stage('Docker Compose Up') {
    steps {
        script {
            def composePath = "${env.WORKSPACE}/docker-compose.yml"
            def envPath = "${env.WORKSPACE}/cicd/.env"
```

```
            def runtimeEnvKeys = [
                'DEARIE_DB_URL', 'DEARIE_DB_USER', 'DEARIE_DB_PASSWORD', 'DEARIE_DB_NAME', 'DEARIE_JWT_SECRET',
                'LIGHT_DB_URL', 'LIGHT_DB_USER', 'LIGHT_DB_PASSWORD', 'LIGHT_DB_NAME', 'LIGHT_JWT_SECRET',
                'KAFKA_BOOTSTRAP_SERVERS', 'KAFKA_TOPIC_NAME', 'KAFKA_CONSUMER_GROUP_ID',
                'OPENAI_API_KEY',
                'S3_ACCESS_KEY', 'S3_SECRET_KEY', 'S3_BUCKET_DEARIE','S3_BUCKET_LIGHTREBORN',
                'NEXT_PUBLIC_NAVER_CLIENT_ID',
                'NEXT_PUBLIC_MAPTILER_KEY', 'NEXT_PUBLIC_MAPBOX_TOKEN'
            ]

            withEnv(generateWithEnvList(runtimeEnvKeys)) {
                sh """
                    docker-compose --env-file ${envPath} -f ${composePath} up -d --build
                """
            }
        }
    }
}

// 5. Flyway 데이터 마이그레이션
stage('Flyway Check and Migration') {
    steps{
        script{

def projects = ['dearie', 'lightreborn']

projects.each { project ->

    def projUpper = project.toUpperCase()
    def networkName = "${project}-net"
    def dbHost = "${project}-db"
    def dbUser = envProps["${projUpper}_DB_USER"] ?: "ssafy"
    def dbPassword = envProps["${projUpper}_DB_PASSWORD"] ?: "ssafy"
    def dbName = project

    // Flyway 스테이지에서
    // 1. Jenkins 컨테이너 내의 SQL 파일 실제 경로
    def sqlPathInJenkinsContainer = "${env.WORKSPACE}/${project}/backend/src/main/resources/db/migration" // (또는 _master)

    // 2. Jenkins 컨테이너의 env.WORKSPACE가 호스트와 매핑된 경로 (추정)
    // 이 부분은 Jenkins 컨테이너 시작 시 설정된 볼륨 매핑에 따라 결정됩니다.
    // 예: env.WORKSPACE가 /var/jenkins_home/workspace/soboro 이고, 이것이 호스트의 /home/ubuntu/jenkins-data/workspace/soboro
    def hostPathToWorkspace = env.WORKSPACE.replaceFirst("^/var/jenkins_home", "/home/ubuntu/jenkins-data") // 이 변환이 실제 호
    def hostSqlPath = "${hostPathToWorkspace}/${project}/backend/src/main/resources/db/migration" // (또는 _master)

    sh """
        echo "Jenkins 컨테이너 내 SQL 경로: ${sqlPathInJenkinsContainer}"
        echo "호스트 머신에서 접근 가능한 SQL 추정 경로: ${hostSqlPath}"

        if [ ! -d "${sqlPathInJenkinsContainer}" ]; then
            echo "⚠️ SQL 파일 경로가 Jenkins 컨테이너 내에 존재하지 않습니다: ${sqlPathInJenkinsContainer}"
            exit 1
        fi
        ls -la "${sqlPathInJenkinsContainer}"

        docker run --rm \\
            --network "${networkName}" \\
            -v "${hostSqlPath}:/flyway/sql" \\
            flyway/flyway \\
            -locations=filesystem:/flyway/sql \\
            -url=jdbc:postgresql://${dbHost}:5432/${dbName} \\
            -user=${dbUser} \\
            -password=${dbPassword} \\
            migrate
        """
        }
    }
  }
}


// 7. 빌드 성공 여부 상태 반영
```

```
    stage('Mark Image Build Success') {
      steps {
        script {
          buildSuccess = true
          echo "🫠 현재 빌드 상태: ${currentBuild.result}"
          echo "✅ 이미지 빌드 성공 상태로 설정: ${buildSuccess}"
        }
      }
    }
  } // 여기에 stages 섹션을 닫는 중괄호 추가

  post {
    always {
      script {
        def sendMessage = { String msg ->
          def payload = groovy.json.JsonOutput.toJson([text: msg])
          writeFile file: 'payload.json', text: payload

          withCredentials([string(credentialsId: MATTERMOST_WEBHOOK_ID, variable: 'MATTERMOST_WEBHOOK')]) {
            sh '''
              curl -X POST -H 'Content-Type: application/json' -d @payload.json $MATTERMOST_WEBHOOK
            '''
          }
        }

        if (buildSuccess || currentBuild.result == 'SUCCESS') {
          sendMessage("🎉 배포 성공 : `${env.ENV}` 환경\n- Job: `${env.JOB_NAME}`\n- Build: #${env.BUILD_NUMBER}")
        } else {
          sendMessage("❌ 배포 실패 : `${env.ENV}` 환경\n- Job: `${env.JOB_NAME}`\n- Build: #${env.BUILD_NUMBER}\n- [로그 확인하기](${er
        }

        // 컨테이너가 안정화된 후에 .env 파일 삭제
        sh """
          echo "🧹 보안상 민감한 파일 정리 중..."
          find . -name ".env" -type f -delete 2>/dev/null || true
          find . -name ".env.production" -type f -delete 2>/dev/null || true
          find . -name ".env.dearie.production" -type f -delete 2>/dev/null || true
          rm -f payload.json 2>/dev/null || true
        """
      }
    }
  }

  success {
    script {
      if (params.ENV == 'master') {
        echo '🎉 Build 성공 → Stable 이미지 태깅 및 푸시'
        sh '''
          # backend
          docker tag dearie-backend dearie-backend:stable
          docker tag lightreborn-backend lightreborn-backend:stable

          # frontend
          docker tag dearie-frontend dearie-frontend:stable
          docker tag lightreborn-frontend lightreborn-frontend:stable

          # build all
          docker build -t dearie-backend:stable .
          docker build -t lightreborn-backend:stable .
          docker build -t dearie-frontend:stable .
          docker build -t lightreborn-frontend:stable .
        '''
      }
    }
  }

  failure {
    script {
      if (params.ENV == 'master') {
        echo '⛔ 실패 → 이전 stable 이미지로 롤백 시도'
        sh '''
          # stop & remove
          docker stop dearie-backend || true
```

```
                    docker stop lightreborn-backend || true
                    docker stop dearie-frontend || true
                    docker stop lightreborn-frontend || true

                    docker rm dearie-backend || true
                    docker rm lightreborn-backend || true
                    docker rm dearie-frontend || true
                    docker rm lightreborn-frontend || true

                    # run rollback
                    docker run -d --name dearie-backend --network dearie-net -p 8082:8082 dearie-backend:stable
                    docker run -d --name lightreborn-backend --network lightreborn-net -p 8081:8081 lightreborn-backend:stable
                    docker run -d --name dearie-frontend --network dearie-net  -p 3001:3001 dearie-frontend:stable
                    docker run -d --name lightreborn-frontend --network lightreborn-net -p 3000:3000 lightreborn-frontend:stable
                '''
            }
        }
    }
}
```

- docker-compose.yml
  - root 폴더에 docker-compose.yml을 위치시킵니다.

```
services:
  # dearie
  backend-dearie:
    build:
      context: ./dearie/backend
    image: dearie-backend:latest
    container_name: dearie-backend
    env_file:
      - ./cicd/.env
    ports:
      - "8082:8082"
    depends_on:
      - redis-dearie
    networks:
      - dearie-net
    environment:
      - SPRING_PROFILES_ACTIVE=prod
      - S3_BUCKET_DEARIE=${S3_BUCKET_DEARIE}

  frontend-dearie:
    build:
      context: ./dearie/frontend
    image: dearie-frontend:latest
    container_name: dearie-frontend
    ports:
      - "3001:3000"
    depends_on:
      - backend-dearie
    networks:
      - dearie-net

  redis-dearie:
    image: redis:latest
    container_name: redis-dearie
    ports:
      - "6380:6379"
    networks:
      - dearie-net

  # lightreborn
  backend-lightreborn:
    build:
      context: ./lightreborn/backend
    image: lightreborn-backend:latest
    container_name: lightreborn-backend
    env_file:
      - ./cicd/.env
    ports:
```

```
    - "8081:8081"
  depends_on:
    - redis-lightreborn
  networks:
    - lightreborn-net
  environment:
    - SPRING_PROFILES_ACTIVE=prod
    - S3_BUCKET_LIGHTREBORN=${S3_BUCKET_LIGHTREBORN}

frontend-lightreborn:
  build:
    context: ./lightreborn/frontend
  image: lightreborn-frontend:latest
  container_name: lightreborn-frontend
  ports:
    - "3000:3000"
  depends_on:
    - backend-lightreborn
  networks:
    - lightreborn-net

redis-lightreborn:
  image: redis:latest
  container_name: redis-lightreborn
  ports:
    - "6379:6379"
  networks:
    - lightreborn-net

networks:
  dearie-net:
    external: true
  lightreborn-net:
    external: true
```

- Dockerfile(Backend)
  - Backend용 Dockerfile을 backend폴더에 위치시킵니다.
    - Lightreborn

```
# 1. 빌드 단계
FROM openjdk:21-jdk-slim AS build

WORKDIR /app

COPY build.gradle settings.gradle ./
COPY gradle ./gradle
COPY gradlew ./
RUN chmod +x gradlew

COPY . .
RUN chmod +x gradlew
RUN ./gradlew build -x test --no-daemon

# 2. 실행 단계 (FFmpeg 포함)
FROM openjdk:21-jdk-slim

WORKDIR /app

# FFmpeg 설치
RUN apt update && \
    apt install -y ffmpeg && \
    apt clean && \
    rm -rf /var/lib/apt/lists/*

ENV FFMPEG_PATH=/usr/bin/ffmpeg

COPY --from=build /app/build/libs/*.jar lightreborn-backend.jar

EXPOSE 8081

CMD ["java", "-jar", "lightreborn-backend.jar"]
```

- dearie

```
# 1. 빌드
FROM openjdk:21-jdk-slim AS build

WORKDIR /app

# Gradle 캐싱을 위해 먼저 의존성 관련 파일 복사
COPY build.gradle settings.gradle ./
COPY gradle ./gradle
COPY gradlew ./

# 실행 권한 부여
RUN chmod +x gradlew

# 전체 프로젝트 복사 후 재빌드
COPY . .
RUN chmod +x gradlew

# 빌드
RUN ./gradlew build -x test --no-daemon

# 실제 런타임 이미지
FROM openjdk:21-jdk-slim

WORKDIR /app

# OpenCV 관련 필수 라이브러리 설치
RUN apt-get update && apt-get install -y --no-install-recommends \
    libopencv-dev \
    libtbb-dev \
    libgtk2.0-dev \
    pkg-config \
    libavcodec-dev \
    libavformat-dev \
    libswscale-dev && \
    rm -rf /var/lib/apt/lists/*

# jar 파일 복사
COPY --from=build /app/build/libs/*.jar dearie-backend.jar

EXPOSE 8082

CMD ["java", "-jar", "dearie-backend.jar"]
```

- Dockerfile(frontend)
  - Frontend용 Dockerfile을 front 폴더에 위치시킵니다.
    - lightreborn

```
# 1. 빌드
FROM node:22-alpine AS build

# 작업 디렉토리 설정
WORKDIR /app

# package.json과 package-lock.json 복사
COPY package.json package-lock.json ./

# 의존성 설치 (package-lock.json을 100% 따름)
RUN npm ci

# env 파일 복사
COPY .env.production .env.production

# 전체 프로젝트 복사
COPY . .

# 빌드 실행
RUN npm run build -- --no-lint
# RUN npm run build
```

```
# 2. 런타임
FROM node:22-alpine

WORKDIR /app

COPY --from=build /app ./

EXPOSE 3000

CMD ["npm", "start"]
```

- dearie

```
# 1. 빌드
FROM node:22-alpine AS build

# 작업 디렉토리 설정
WORKDIR /app

# package.json과 package-lock.json 복사
COPY package.json package-lock.json ./

# 의존성 설치 (package-lock.json을 100% 따름)
RUN npm ci

# env 파일 복사 - dearie의 env 어떻게 할지
COPY .env.dearie.production .env.production

# 전체 프로젝트 복사
COPY . .

# 빌드 실행
RUN npm run build -- --no-lint

# 2. 런타임
FROM node:22-alpine

WORKDIR /app

COPY --from=build /app ./

EXPOSE 3001

CMD ["npm", "start"]
```

- dearie/frontend/next.config.mjs
  - 한 서버에 두 프로젝트를 띄우기 때문에 basePath에 `/dearie`를 추가해줍니다.
  - PWA 설정을 추가합니다.

```
basePath: '/dearie',
 // PWA 관련 헤더 설정
 async headers() {
  return [
   {
    source: '/dearie/sw.js',
    headers: [
     {
      key: 'Cache-Control',
      value: 'public, max-age=0, must-revalidate',
     },
     {
      key: 'Service-Worker-Allowed',
      value: '/dearie',
     },
    ],
   },
   {
    source: '/dearie/manifest.json',
    headers: [
     {
      key: 'Content-Type',
```

```
          value: 'application/manifest+json',
        },
        {
          key: 'Cache-Control',
          value: 'public, max-age=0, must-revalidate',
        },
      ],
    },
  ];
},
```

네트워크 설정

- 백엔드, 프론트엔드, nginx, jenkins간 원활한 연결을 위해 network를 생성합니다.

```
docker network create bridge # jenkins
docker network create dearie-net
docker network create lightreborn-net
```

- 생성한 네트워크 중 연결될 데이터 간 컨테이너끼리 연결해줍니다.

```
docker netowork connect dearie-net dearie-db
docker netowork connect lightreborn-net lightreborn-db
```

덤프파일 최신본

- lightreborn

  V5__data_init.sql

- dearie

  V8__data_init.sql